

A method for scheduling in parallel manufacturing systems with flexible resources

SIGURDUR ÓLAFSSON¹ and LEYUAN SHI²

¹Department of Industrial and Manufacturing Systems Engineering, 205 Engineering Annex, Iowa State University, Ames, IA 50011, USA

E-mail: olafsson@iastate.edu

²Department of Industrial Engineering, University of Wisconsin-Madison, Madison, WI 53706, USA

E-mail: leyuanshi@ie.engr.wisc.edu

Received November 1997 and accepted November 1998

We address the Parallel-Machine Flexible-Resource Scheduling (PMFRS) problem of simultaneously allocating flexible resources, and sequencing jobs, in cellular manufacturing systems where the cells are configured in parallel. We present a new solution methodology for the PMFRS problem called the Nested Partitions (NP) method. This method combines global sampling of the feasible region and local search heuristics. To efficiently apply the NP method we reformulate the PMFRS problem, develop a new sampling algorithm that can be used to obtain good feasible schedules, and suggest a new improvement heuristic. Numerical examples are also presented to illustrate the new method.

1. Introduction

The scheduling function plays an important role in many cellular manufacturing systems. Such scheduling is often performed along several dimensions, including assigning jobs to manufacturing cells, sequencing the jobs within each cell, and allocating resources to the cells. Sometimes two or more of these scheduling decisions must be made simultaneously. An example of this could be an assembly system where several product families are assembled. If requirements vary drastically from one family to another each cell may be uniquely configured to produce specific families and each family must therefore be processed in a given cell. On the other hand, workers may be cross-trained and could be dynamically assigned to any of the cells. The scheduling problem is therefore to simultaneously sequence the jobs within each cell, determine how many workers should be assigned to each cell at each time, and find a starting time for each job.

Scheduling for parallel manufacturing systems has traditionally focused on two main issues: assigning jobs to machines and sequencing the jobs that are assigned to the same machine [1,2]. A few studies have focused on scheduling, either job assignment or sequencing, when the speed of the machines is variable, such as is the case for many tooling machines [3–5]. On the other hand, in cellular manufacturing systems the processing speed may depend on some flexible resource, such as skilled labor, that is competed for by the cells. The problem of simul-

taneously allocating these flexible resources to the cells and scheduling the jobs within each cell has received relatively little attention, and these two problems have traditionally been considered separately. For example, Frenk *et al.* [6] and Boxma *et al.* [7] consider resource allocation in manufacturing systems, whereas So [8] considers the problem of finding a feasible schedule for each machine in a parallel manufacturing system. The simultaneous job scheduling and resource allocation was first discussed by Daniels and Mazzola [9]. The two problems can be seen to be intimately related and considering them simultaneously may produce significant benefits [10,11]. In related work, joint sequencing and resource allocation decisions have also been investigated for single machine shop scheduling [12–14]. However, in this context the resources are necessarily non-renewable and there is no competition for resources across manufacturing cells.

In this paper we introduce a new methodology for scheduling on parallel manufacturing cells: the Nested Partitions (NP) method [15]. The NP method is a flexible method for combinatorial optimization that combines global and local search. We first apply the NP method to the Parallel-Machine Flexible-Resource Scheduling (PMFRS) problem as originally formulated by Daniels *et al.* [11]. We then reformulate the problem and use this formulation to develop a more efficient implementation of the NP method. This latter implementation incorporates the special structure of the PMFRS problem into

both the global and local search phases. The remainder of the paper is organized as follows. In Section 2 we review the PMFRS problem, and in Section 3 we develop a generic implementation of the NP method. In Section 4 we reformulate the PMFRS problem, and in Section 5 we describe a more efficient NP algorithm based on the reformulated problem. In Section 6 numerical examples are presented, and Section 7 contains some concluding remarks and plans for future research.

2. The PMFRS problem

A cellular manufacturing system has a set of jobs, indexed by $i \in \mathcal{N} = \{1, 2, \dots, n\}$, to be processed in one of m manufacturing cells, indexed by $j \in \mathcal{M} = \{1, 2, \dots, m\}$. There is a fixed amount R of renewable resources that are flexible. The set \mathcal{N}_j of jobs processed in cell $j \in \mathcal{M}$ is fixed, but the sequence of jobs within a cell, and the amount of flexible resources used in each cell at each time can be varied. The objective is to minimize the makespan $C = \max_{i \in \mathcal{N}} C_i$, where C_i is the completion time of job $i \in \mathcal{N}$. This objective is often of interesting for parallel systems, in particular since schedules with low makespan tend to balance the load between cells [2].

The optimization problem can be seen to be threefold. First a dynamic resource allocation must be determined. Second a sequence in which jobs should be processed within each cell must be determined, and third a starting time for each job must be found. This scheduling problem has previously been discussed in Daniels *et al.* [11] where it is termed the Parallel-Machine Flexible-Resource Scheduling (PMFRS) problem. The decision variables are defined as

$$y_{ih} = \begin{cases} 1 & \text{if job } i \text{ precedes job } h, \text{ where } i, h \in \mathcal{N}_j, \\ & j \in \mathcal{M}, \\ 0 & \text{otherwise.} \end{cases}$$

$$x_{irt} = \begin{cases} 1 & \text{if job } i \text{ completes processing with } r \text{ resources} \\ & \text{at time } t, \\ 0 & \text{otherwise.} \end{cases}$$

We let \hat{p}_{ir} denote the service time of job $i \in \mathcal{N}$ when $r \leq R$ resources are allocated to i -th job and p_i denote the actual service time of a job $i \in \mathcal{N}$. We also let T be an upper bound on the makespan, for example $T = \max_{j \in \mathcal{M}} \sum_{i \in \mathcal{N}_j} \hat{p}_{i1}$. In terms of the decision variables $\mathbf{x} = \{x_{irt}\}$ and $\mathbf{y} = \{y_{ih}\}$, the PMFRS optimization problem can be formulated as the following 0–1 Integer Programming (IP) problem.

$$\min_{\mathbf{x}, \mathbf{y}} C \quad (1)$$

s.t.

$$\sum_{r=1}^R \hat{p}_{ir} \sum_{t=1}^T x_{irt} = p_i, \quad i \in \mathcal{N}, \quad (2)$$

$$\sum_{t=1}^T t \left(\sum_{r=1}^R x_{irt} \right) = C_i, \quad i \in \mathcal{N}, \quad (3)$$

$$\sum_{r=1}^R \sum_{t=1}^T x_{irt} = 1, \quad i \in \mathcal{N}, \quad (4)$$

$$C_i \leq C, \quad i \in \mathcal{N}, \quad (5)$$

$$C_h - C_i + T(1 - y_{ih}) \geq p_h, \quad i \in \mathcal{N}_j, h \in \mathcal{N}_j \setminus \{i\}, j \in \mathcal{M}, \quad (6)$$

$$y_{ih} + y_{hi} = 1, \quad i \in \mathcal{N}_j, h \in \mathcal{N}_j \setminus \{1, 2, \dots, i\}, j \in \mathcal{M}, \quad (7)$$

$$\sum_{i \in \mathcal{N}} \sum_{r=1}^R \sum_{t=1}^{t+\hat{p}_{ir}-1} r \times x_{irt} \leq R, \quad t \in \{1, 2, \dots, T\}, \quad (8)$$

$$y_{ih}, x_{irt} \in \{0, 1\}, \quad 1 \leq r \leq R, \quad i \in \mathcal{N}_j, h \in \mathcal{N}_j \setminus \{i\}, \\ j \in \mathcal{M}, t \in \{1, 2, \dots, T\}, \quad (9)$$

$$C_i \geq 0, \quad i \in \mathcal{N}. \quad (10)$$

Daniels *et al.* [11] have shown how this problem can be solved using a branch-and-bound algorithm and have developed a heuristic that is based on its static equivalent. They show that the problem is NP-hard, and solve it exactly using the branch-and-bound method for relatively small problem instances, and for larger problems using the heuristic. This heuristic is based on the static equivalent of the PMFRS problem, where the resources are not flexible and stay in the same cell for the entire time horizon. It is readily seen that for the static problem the sequence of jobs does not matter and the problem reduces to a simple resource allocation problem. This problem can be solved efficiently to optimality [11], and is therefore a convenient benchmark to evaluate the performance of algorithms for the PMFRS problem, as well as to evaluate the benefits of flexible resources.

In this paper we present an alternative approach to solving the PMFRS problem: the Nested Partitions (NP) method. As we will see, a critical feature of the NP method is that although it partitions the feasible region in a similar fashion to the branch-and-bound method, it only needs to keep track of a limited number of branches in each iteration. Hence it is applicable even for very large problems and may be useful when the branch-and-bound method experiences memory problems. Furthermore, finite time convergence of the NP method is guaranteed so that it finds the optimal schedule if enough computational effort is used. The method is very flexible and is capable of incorporating efficient heuristics to speed convergence and to rapidly obtain good solutions. The NP method is therefore an attractive alternative to the previously proposed branch-and-bound and heuristic methods.

3. The nested partitions method for the PMFRS problem

In this section we describe the new Nested Partitions (NP) method and present a simple implementation of the method for the PMFRS problem. The generic NP algorithm has been discussed before [15], but for completeness a brief description is included below.

3.1. The NP methodology

The NP method may be considered to be an adaptive sampling algorithm that uniquely combines local and global search. If Θ denotes the feasible region, then in the global search phase all of Θ is sampled using a random sampling scheme that concentrates the sampling in subregions that appear to be the most promising. The sampling distribution is adapted using *partitioning* that makes the sampling more concentrated, and *backtracking* that makes it less concentrated. The partitioning of the feasible region proceeds similarly to the branching in the branch-and-bound method and creates a space of subsets of Θ that we call Σ . If, for example, the feasible region contains four points $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_4\}$, it can first be partitioned into two subregions $\{\theta_1, \theta_2\}$ and $\{\theta_3, \theta_4\}$; which can then be partitioned further into four singleton subregions $\{\theta_1\}$, $\{\theta_2\}$, $\{\theta_3\}$, and $\{\theta_4\}$. Hence $\Sigma = \{\{\theta_1, \theta_2, \theta_3, \theta_4\}, \{\theta_1, \theta_2\}, \{\theta_3, \theta_4\}, \{\theta_1\}, \{\theta_2\}, \{\theta_3\}, \{\theta_4\}\}$. These subregions may be considered to form a partitioning tree and we define the *depth* $d: \Sigma \rightarrow \mathbf{R}$ of each region to be its depth in this partitioning tree. Thus for our simple example, $d(\{\theta_1, \theta_2, \theta_3, \theta_4\}) = 0$, $d(\{\theta_1, \theta_2\}) = 1$, and $d(\{\theta_3\}) = 2$.

The NP method always considers a subregion $\sigma(k) \in \Sigma$ to be the most promising in the k -th iteration and moves iteratively between elements of Σ using partitioning and backtracking. The partitioning corresponds to moving to a region of greater depth and the backtracking implies that the algorithm moves to region of less depth. For the four point example, if $\sigma(k) = \{\theta_3\}$ is the most promising region in the k th iteration and backtracking is found to be the appropriate move, then $\sigma(k+1) = \{\theta_3, \theta_4\}$, or $\sigma(k+1) = \{\theta_1, \theta_2, \theta_3, \theta_4\}$ in the next iteration.

In the k th iteration the sampling is concentrated in the most promising region $\sigma(k)$ but the remainder of the feasible region $\Theta \setminus \sigma(k)$, called the surrounding region, is also sampled but with less intensity. The local search phase starts with the sample points from each subregion and uses local optimization methods or heuristics to find better neighboring points. The local search information is incorporated into a *promising index* function, $I: \Sigma \rightarrow \mathbf{R}$, that is used to determine how the global search is concentrated in the next iteration. Therefore, each iteration of the NP method consists of four main steps:

- (i) *Partitioning*. The first step of each iteration is to partition the current most promising region

$\sigma(k) \in \Sigma$ into Q subregions $\{\sigma_q(k)\}_{q=1}^Q$ and aggregate the surrounding region into one region $\Theta \setminus \sigma(k)$. The partitioning strategy imposes a structure on the feasible region and if most of the good schedules tend to be clustered together in the same subregions, it is likely that the algorithm quickly concentrates the search in these subsets of the feasible region.

- (ii) *Random sampling*. The next step is to randomly sample from each of the subregions and the aggregated surrounding region. This can be done in almost any fashion. The only condition is that each schedule should be selected with a positive probability. Uniform sampling can always be chosen, but it may often be worthwhile to incorporate special structure.
- (iii) *Estimating the promising index*. Once each region has been sampled the next step is to use the sample points to estimate the promising index of each region. The only requirement imposed on a promising index is that it agrees with the original performance function on regions of maximum depth, i.e., on singletons.
- (iv) *Backtracking*. If one of the subregions has the best estimated promising index the algorithm moves to this region and considers it the most promising region in the next iteration. If the surrounding region has the best estimated promising index it backtracks to a larger region.

The NP method has been shown to converge in finite time to the global optimum of problems such as the PMFRS problem and other combinatorial problems [15]. The basic idea of the convergence analysis is to observe that the sequence $\{\sigma(k)\}_{k=0}^{\infty}$ of most promising regions is an absorbing Markov chain with state space Σ . Furthermore, the set of absorbing states are all singletons, and corresponds exactly to the set of global optima. Therefore, the Markov chain is absorbed at a global optimum, and since the state space is finite it occurs in finite time.

3.2. An NP algorithm for the PMFRS problem

Since the PMFRS problem can be formulated as a 0–1 IP problem, and it is well known that branch-and-bound can be used to solve such problem, a natural method of partitioning is to follow the branching procedure typically used for such problems. If this approach is used, a region $\sigma \in \Sigma$ of depth $d(\sigma)$ corresponds to a branch whose root node is of depth $d(\sigma)$. The schedules in σ correspond to the leaf nodes of the branch. The promising index $I: \Sigma \rightarrow \mathbf{R}$ can be a summary statistic, such as the best makespan,

$$I(\sigma) = \min_{(\mathbf{x}, \mathbf{y}) \in \sigma} C(\mathbf{x}, \mathbf{y}), \quad \sigma \in \Sigma.$$

This promising index function is naturally estimated by the best makespan among all the schedules sampled from the region. Backtracking is also simple and we can consider two alternatives: backtracking to the next larger region, and backtracking to the entire feasible region. Continuing with the analogy to branch-and-bound, the first backtracking rule corresponds to moving to a branch whose root node is of one less depth than the root node of the current most promising region (branch), and the second backtracking rule corresponds to moving to the entire tree.

The description above stresses the similarities between the NP method and the branch-and-bound method. It is therefore appropriate to also highlight the differences. The properties of the branch-and-bound method are well known, so we focus on the potential advantages of the NP method. These are primarily twofold. First, the branch-and-bound requires upper- and lower bounds on each branch and these bounds are used to systematically fathom the branches. The NP method does not require a lower bound, but rather the search is guided by the estimated promising index of the region, which is an upper bound on the performance. This may be particularly advantageous when estimating the lower bounds is difficult and computationally intensive. Secondly, the branch-and-bound method may encounter memory problems in maintaining the branches that have not yet been fathomed. The NP method, however, only keeps track of the subregions of the current most promising region. This is typically a much smaller number and may be advantageous for large problems.

The NP algorithm described above is simple and has nice analogies to the familiar branch-and-bound method. It is completely generic in the sense that it can be applied to any IP problem. However, by treating the PMFRS problem as a generic 0–1 IP problem we ignore its special structure, and this may be useful in rapidly finding the optimal schedule. We therefore aim at developing an efficient implementation of the NP method that takes maximum advantage of the structure of the PMFRS problem. To this end we start by reformulating the problem.

4. An alternative formulation of the PMFRS problem

We define a vector $\mathbf{R} = \{R_i\}_{i \in \mathcal{N}}$ to denote the resources allocated to each job and a vector $\mathbf{T} = \{T_i\}_{i \in \mathcal{N}}$ to denote the starting time of each job. Specifying these two vectors completely determines a schedule $\theta = (\mathbf{R}, \mathbf{T})$. In addition to being integer valued we assume the following constraints on $\mathbf{R} \in \mathbf{N}^n$ and $\mathbf{T} \in \mathbf{N}^n$. At each time no more than R resources are allocated to the jobs currently being processed. Furthermore, for each job $i \in \mathcal{N}$, changing the starting time T_i to $T_i - 1$ violates the resource restrictions. This means that each job is scheduled as early as it does

not interfere with any other jobs. Schedules with this property are often referred to as active schedules [2]. Lastly, only one job is processed at a time in each cell.

This reformulated PMFRS optimization problem can be stated mathematically as follows.

$$\min_{\theta=(\mathbf{R},\mathbf{T})} C(\theta), \quad (11)$$

s.t.

$$\sum_{i=1}^n R_i \times \chi_{\{T_i, \dots, T_i + \hat{p}_{R_i}\}}(t) \leq R, \quad t = 1, 2, \dots, T, \quad (12)$$

$$\sum_{i=1}^n R_i \times \chi_{\{T_i, \dots, T_i + \hat{p}_{R_i}\}}(T_i - 1) > R - R_i, \quad \tilde{i} \in \mathcal{N}, \quad (13)$$

$$\sum_{i \in \mathcal{N}_j} \chi_{\{T_i, \dots, T_i + \hat{p}_{R_i}\}}(t) \leq 1, \quad t = 1, 2, \dots, T, \quad j \in \mathcal{M}, \quad (14)$$

$$\mathbf{R}, \mathbf{T} \in \mathbf{N}^n. \quad (15)$$

Here $\chi_A(\cdot)$ is the indicator function, $\chi_A(t) = 1$ if $t \in A$ and $\chi_A(t) = 0$ if $t \notin A$. The constraint (12) ensures that the available amount of resources is not exceeded. The constraint (13) ensures that all schedules are active. Constraint (14) ensures that only one job in each cell is scheduled at the same time. The feasible region is given by $\Theta = \{(\mathbf{R}, \mathbf{T}) \in \mathbf{N}^n \times \mathbf{N}^n \mid \text{Equations (12)–(14) hold}\}$, and a schedule $\theta \in \Theta$ may be written as $\theta = \{(R_1, T_1), (R_2, T_2), \dots, (R_n, T_n)\}$.

Example 1. As an illustration, consider the simple example shown in Fig. 1a–c. This system has $m = 2$ manufacturing cells, $n = 3$ jobs, and $R = 2$ flexible resources. The schedule in Fig. 1a is feasible. It allocates all of the flexible resources to each job while being processed and the two cells are therefore never in operation at the same time. The schedule in Fig. 1b divides the resource between the second job in the first cell and the job in the second cell. These jobs are therefore processed in parallel after the first job is completed. This schedule is also feasible. The schedule in Fig. 1c is infeasible. This is because starting the second job of the first cell earlier would not violate the resource constraint, so this schedule violates constraint (13). We note that this schedule would not be infeasible according to the original formulation but that nothing is lost by making such schedules infeasible.

As illustrated in the above example the new formulation of the PMFRS reduces the feasible region. However,

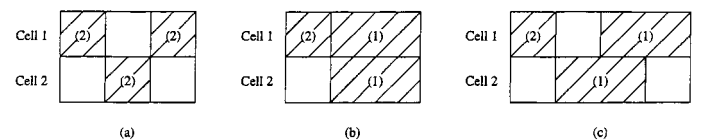


Fig. 1. Possible schedules; the shaded areas indicate jobs whilst the number of resources are given in brackets.

the optimal schedule(s) are retained as is established in the next theorem.

Theorem 1. For any feasible solution to Equations (13)–(15) there is a corresponding solution to Equations (2)–(10) that has exactly the same makespan. Conversely, for any feasible solution to Equations (2)–(10) there is a corresponding solution to Equations (13)–(15) that has makespan that is less than or equal to the makespan of the first solution.

Proof. (a) First assume that a schedule $\theta = (\mathbf{R}, \mathbf{T})$ satisfies Equations (13)–(15). Define the equivalent decision variables

$$y_{ih} = \begin{cases} 1 & T_i < T_h, \quad i, h \in \mathcal{N}, \\ 0 & \text{otherwise.} \end{cases}$$

$$x_{irt} = \begin{cases} 1 & r = R_i, \quad t = T_i + \hat{p}_{iR_i}, \quad i \in \mathcal{N}, \\ 0 & \text{otherwise.} \end{cases}$$

First we must show that these decision variables are feasible for the original formulation, that is, that they satisfy the constraints (2)–(10). Note that for all $i \in \mathcal{N}$

$$\sum_{r=1}^R \hat{p}_{ik} \sum_{t=1}^T x_{irt} = \sum_{r=1}^R \hat{p}_{ik} \chi_{\{R_i\}}(r) = \hat{p}_{iR_i} = p_i.$$

Therefore Equation (2) holds. Similarly, for all $i \in \mathcal{N}$

$$\sum_{t=1}^T t \left(\sum_{r=1}^R x_{irt} \right) = \sum_{t=1}^T t \chi_{\{T_i + \hat{p}_{iR_i}\}}(t) = T_i + \hat{p}_{iR_i} = C_i.$$

The constraints (4) and (5) hold trivially. Now let $j \in \mathcal{M}$, $i \in \mathcal{N}$ and $h \in \mathcal{N} \setminus \{i\}$. If $y_{ih} = 0$ then Equation (6) is trivially satisfied. If $y_{ih} = 1$ then $T_i < T_h$ by definition and

$$C_h - C_i + T(1 - y_{ih}) = C_h - C_i = T_h + \hat{p}_{hR_h} - T_i + \hat{p}_{iR_i} \\ \geq \hat{p}_{hR_h} - \hat{p}_{iR_i} = p_h - p_i \geq p_h,$$

so the constraint (6) holds. Now for any cell $j \in \mathcal{M}$ and jobs in that cell $i \in \mathcal{N}$ and $h \in \mathcal{N} \setminus \{i\}$, then by equation (14), either $T_i < T_h$ or $T_i > T_h$. Hence either $y_{ih} = 1$ and $y_{hi} = 0$, or $y_{ih} = 0$ and $y_{hi} = 1$. Consequently $y_{ih} + y_{hi} = 1$ and Equation (7) holds. Now for any $t \in \{1, 2, \dots, T\}$

$$\sum_{i \in \mathcal{N}} \sum_{r=1}^R \sum_{l=t}^{t+\hat{p}_{ir}-1} r \times x_{irl} = \sum_{i \in \mathcal{N}} \sum_{l=t}^{t+\hat{p}_{ir}-1} \sum_{r=1}^R r \times x_{irl} \\ = \sum_{i \in \mathcal{N}} \sum_{l=t}^{t+\hat{p}_{ir}-1} R_i \chi_{\{T_i + \hat{p}_{iR_i}\}}(l) \\ = \sum_{i \in \mathcal{N}} R_i \sum_{l=t}^{t+\hat{p}_{ir}-1} \chi_{\{T_i + \hat{p}_{iR_i}\}}(l) \\ = \sum_{i \in \mathcal{N}} R_i \times \chi_{\{T_i, \dots, T_i + \hat{p}_{iR_i}\}}(t) \\ \leq R.$$

This shows that constraint (8) holds. Equations (9) and (10) hold trivially. The feasibility of the solution has now been established. The fact that (\mathbf{x}, \mathbf{y}) and (\mathbf{R}, \mathbf{T}) have the same makespan is trivial.

(b) Now assume that we start with a feasible solution (\mathbf{x}, \mathbf{y}) to original problem. A feasible schedule $\theta = (\mathbf{R}, \mathbf{T})$, that has makespan less than or equal to the makespan corresponding to (\mathbf{x}, \mathbf{y}) , can be constructed as follows. Let the resource vector be defined as

$$R_i = \sum_{t=1}^T \sum_{r=1}^R r \times x_{irt}, \quad i \in \mathcal{N}. \quad (16)$$

Then the starting times can be obtained iteratively.

Step 0. Let $j = 1$ and $\mathcal{N}_s = \emptyset$.

Step 1. Let $\mathcal{N}_j^U = \mathcal{N}_j$ and $t_0 = 1$.

Step 2. Select $i \in \{h \in \mathcal{N}_j^U : y_{hg} = 1, \forall g \in \mathcal{N}_j^U \setminus \{h\}\}$.

Step 3. Let

$$T_i = \min \left\{ t \geq t_0 : R - \sum_{h \in \mathcal{N}_s} R_h \times \chi_{\{T_h, T_h+1, \dots, T_h + \hat{p}_{hR_h}\}}(u) \geq R_i, u = t, \dots, t + \hat{p}_{iR_i} \right\}.$$

Let $t_0 = T_i + \hat{p}_{iR_i} + 1$.

Step 4. Let $\mathcal{N}_j^U = \mathcal{N}_j^U \setminus \{i\}$ and $\mathcal{N}_s = \mathcal{N}_s \cup \{i\}$.

If $\mathcal{N}_j^U \neq \emptyset$ then go back to Step 2. Otherwise continue to Step 5.

Step 5. If $j < m$, let $j = j + 1$ and go back to Step 1. Otherwise stop.

It is clear that this construction maintains feasibility, that is, (\mathbf{R}, \mathbf{T}) satisfies Equations (12)–(15). Therefore, all that remains is to show that the makespan corresponding to $\theta = (\mathbf{R}, \mathbf{T})$ is less than or equal to the makespan corresponding to the solution (\mathbf{x}, \mathbf{y}) of the original problem, that is,

$$\max_{j \in \mathcal{M}} \max_{h \in \mathcal{N}_j} \left\{ \sum_{r=1}^R \sum_{t=1}^T t \times x_{hrt} \right\} \geq \max_{j \in \mathcal{M}} \max_{h \in \mathcal{N}_j} \{ T_h + \hat{p}_{hR_h} \}. \quad (17)$$

We will prove this by contradiction. Assume that Equation (17) does not hold. Then there exists at least one cell $j' \in \mathcal{M}$ such that

$$\max_{h \in \mathcal{N}_{j'}} \left\{ \sum_{r=1}^R \sum_{t=1}^T t \times x_{hrt} \right\} < \max_{h \in \mathcal{N}_{j'}} \{ T_h + \hat{p}_{hR_h} \}.$$

Furthermore, since the construction algorithm preserves the original sequencing these maxima are realized for the same job $i' \in \mathcal{N}_{j'}$

$$\sum_{r=1}^R \sum_{t=1}^T t \times x_{i'rt} < T_{i'} + \hat{p}_{i'R_{i'}}.$$

However, since (\mathbf{x}, \mathbf{y}) is feasible then $T_i^* = \sum_{r=1}^R \sum_{t=1}^T t \times x_{i'rt} - \hat{p}_{i'R_i}$ is a feasible starting time for job $i' \in \mathcal{N}_j$ and $T_i^* < T_i$. This violates constraint (13) and is therefore a contradiction. ■

This theorem shows that the optimal solution to the PMFRS problem is an active schedule, so all the optimal schedules are contained in the reduced feasible region defined by Equations (11)–(15) above.

5. Improved NP algorithm

We now describe how the NP method can be implemented for the reformulated PMFRS problem defined by Equations (11)–(15) above. This implementation takes advantage of the special structure of the PMFRS problem both in the global sampling and the local search.

5.1. Partitioning

First we address the partitioning. The basic idea is to completely schedule one job at each level of the partitioning tree so that a region of depth d is defined by d jobs being scheduled, or fixed, and the remaining $n - d$ jobs being free. By Equation (13) each job is scheduled as early as possible given all other jobs being fixed. Accordingly, we adopt the rule of scheduling each job as early as possible given all the jobs that have already been fixed. Therefore, selecting a job to be fixed early gives the job certain priority. In other words, jobs that are assigned resources and starting time at low depth tend to be scheduled earlier than jobs that are fixed at greater depth. This imposes a structure on the feasible region and allows for special structure to be incorporated into both the global and local search phases.

Recall that $\theta \in \Theta$ can be represented as $\theta = (\mathbf{R}, \mathbf{T}) = \{(R_1, T_1), (R_2, T_2), \dots, (R_n, T_n)\}$, and the partitioning fixes one of these pairs (R_{i_d}, T_{i_d}) to a given value at depth d in the partitioning tree, $d = 1, 2, \dots, n$. The depth one subregions are therefore defined by the following $R \times n$ sets.

$$\begin{aligned} &\{(r, 1), (\cdot, \cdot), \dots, (\cdot, \cdot)\}, \quad r \in \{1, 2, \dots, R\}, \\ &\{(\cdot, \cdot), (r, 1), \dots, (\cdot, \cdot)\}, \quad r \in \{1, 2, \dots, R\}, \\ &\quad \vdots \\ &\{(\cdot, \cdot), (\cdot, \cdot), \dots, (r, 1)\}, \quad r \in \{1, 2, \dots, R\}. \end{aligned}$$

The depth two regions are similarly defined by fixing two elements and so forth.

Given a region $\sigma \in \Sigma$, that determines $d(\sigma)$ of these pairs, we let $\mathbf{R}(\sigma)$ denote the levels of resources and $\mathbf{T}(\sigma)$ the starting times for the $d(\sigma)$ jobs that have already been scheduled. To describe the partitioning procedure in detail it is convenient to establish some more notation. We let $\mathcal{N}_j^U(\sigma)$ denote the set of unscheduled jobs in cell $j \in \mathcal{M}$ and $\mathcal{N}^U(\sigma) = \cup_{j \in \mathcal{M}} \mathcal{N}_j^U(\sigma)$ denote all the

unscheduled jobs. We let $t_j^0(\sigma)$ denote the first possible starting time in cell j ,

$$t_j^0(\sigma) = \max_{i \in \mathcal{N}_j \setminus \mathcal{N}_j^U(\sigma)} \{T_i(\sigma) + \hat{p}_{iR_i(\sigma)} + 1\}.$$

Finally, we let the vector $\mathbf{R}^A(\sigma) = \{R_t^A(\sigma)\}_{t=1,2,\dots,T}$ denote the available resources, i.e., the resources that have not been allocated, at each time. In terms of (\mathbf{R}, \mathbf{T}) it can be written as

$$R_t^A(\sigma) = R - \sum_{i=1}^n R_i(\sigma) \times \chi_{\{T_i, \dots, T_i + \hat{p}_{iR_i(\sigma)}\}}(t).$$

We can now describe the partitioning as follows. Assume we want to partition a region $\sigma(k) \in \Sigma$. We let each subregion correspond to scheduling one of the jobs in $\mathcal{N}^U(\sigma(k))$ next with a given amount of resources. Each subregion is otherwise identical to $\sigma(k)$. Assume that job $i \in \mathcal{N}_j^U(\sigma(k))$ determines a subregion $\sigma_l(k)$ with $r \in \{1, 2, \dots, R\}$ resources allocated to this job. Then calculate its starting time as the earliest feasible starting time

$$T_i(\sigma_l(k)) = \arg \min_{t \geq t_j^0(\sigma(k))} \{R_t^A(\sigma(k)) \geq r\}. \quad (18)$$

Generate a processing time $p_i = \hat{p}_{ir}$ for the job, and update the earliest feasible starting time in the cell for the subregion, $t_j^0(\sigma_l(k)) = T_i(\sigma_l(k)) + p_i + 1$. The earliest feasible starting time for all other cells is the same for $\sigma_l(k)$ as $\sigma(k)$. Finally, reduce the amount of available resources for the time job i is being processed, that is, let $R_t^A(\sigma_l(k)) = R_t^A(\sigma_l(k)) - r$ for all t such that $T_i(\sigma_l(k)) \leq t \leq T_i(\sigma_l(k)) + p_i$. This completely defines the subregion.

Example 2. Figure 2 partially illustrates the partitioning tree for the simple problem of Example 1 above. Recall that this example has $n = 3$ jobs to be processed on one of

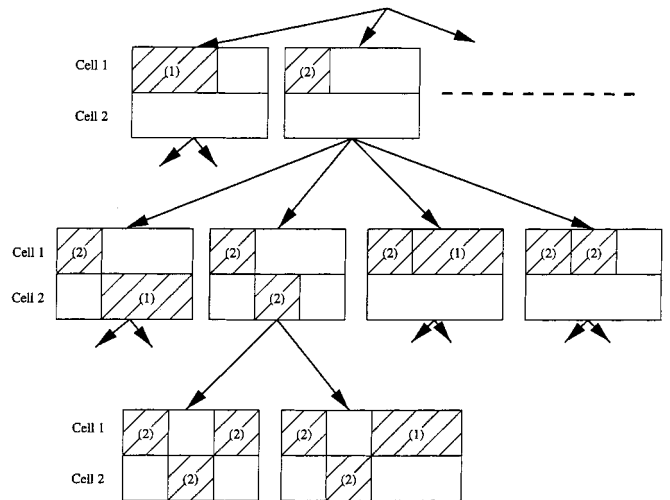


Fig. 2. Part of a partitioning tree.

$m = 2$ cells which have $R = 2$ flexible resources at their disposal. At depth one there are $R \times \sum_{j \in \mathcal{M}} |N_j^U| = 2 \times (2 + 1) = 6$ subregions, two of which are shown in Fig. 2. We assume that the jobs to be processed in cell 1 are labeled as job 1 and 2, and that the job in cell 2 is labeled as job 3. We assume that all the jobs are identical with a processing time of one time unit if they have two resources, and two time units if they have only one resource. The six depth one subregions can therefore be represented as

$$\begin{aligned} \sigma_1(0) &= \{(1, 1), (\cdot, \cdot), (\cdot, \cdot)\}, & \sigma_2(0) &= \{(2, 1), (\cdot, \cdot), (\cdot, \cdot)\}, \\ \sigma_3(0) &= \{(\cdot, \cdot), (1, 1), (\cdot, \cdot)\}, & \sigma_4(0) &= \{(\cdot, \cdot), (2, 1), (\cdot, \cdot)\}, \\ \sigma_5(0) &= \{(\cdot, \cdot), (\cdot, \cdot), (1, 1)\}, & \sigma_6(0) &= \{(\cdot, \cdot), (\cdot, \cdot), (2, 1)\}. \end{aligned}$$

Now if in the next iteration $\sigma(1) = \sigma_1(0) = \{(2, 1), (\cdot, \cdot), (\cdot, \cdot)\}$ then there are four subregions

$$\begin{aligned} \sigma_1(1) &= \{(2, 1), (\cdot, \cdot), (1, 2)\}, & \sigma_2(1) &= \{(2, 1), (\cdot, \cdot), (2, 2)\}, \\ \sigma_3(1) &= \{(2, 1), (1, 2), (\cdot, \cdot)\}, & \sigma_4(1) &= \{(2, 1), (2, 2), (\cdot, \cdot)\}. \end{aligned}$$

Now if, as is illustrated in Fig. 2, the next most promising region is $\sigma(2) = \{(2, 1), (\cdot, \cdot), (2, 2)\}$ then there are only two subregions

$$\begin{aligned} \sigma_1(2) &= \{(2, 1), (1, 3), (2, 2)\}, \\ \sigma_2(2) &= \{(2, 1), (2, 3), (2, 2)\}. \end{aligned}$$

These subregions are now singletons and completely define a feasible schedule.

The next proposition establishes that this is in fact a valid method of partitioning.

Theorem 2. *Every feasible schedule, i.e., point in Θ , corresponds to a maximum depth region obtained by applying the above partitioning procedure. Conversely, every maximum depth region corresponds to a point in Θ .*

Proof. Let $\theta \in \Theta$ and write $\theta = (\mathbf{R}, \mathbf{T})$. We need to identify a maximum depth region that corresponds to this point. The partitioning procedure sequentially selects the jobs and schedules them at the first feasible point. Hence, given θ , we need to construct a sequence of jobs i_1, i_2, \dots, i_n with the following property. Assume that job i_l is given resources R_{i_l} , $l = 1, 2, \dots, n$. If jobs i_1, \dots, i_l have already been scheduled, then the earliest feasible time for job i_{l+1} to start is $T_{i_{l+1}}$, $l = 1, 2, \dots, n - 1$. We can generate this sequence as follows. Initialize by setting the list of scheduled jobs to be empty, $\mathcal{N}_s = \emptyset$. In the first iteration, select a job that has the first possible starting time, i.e., select $i_1 \in \arg \min_{i \in \mathcal{N} \setminus \mathcal{N}_s} T_i$, and let $\mathcal{N}_s = \mathcal{N}_s \cup \{i_1\}$. If more than one are started at the same time, one may be selected arbitrarily. Now simply repeat the process, i.e., select a job $i_2 \in \arg \min_{i \in \mathcal{N} \setminus \mathcal{N}_s} T_i$. This is repeated until there are no more jobs. We use induction to see that this sequence has the right properties. First consider job i_1 . The earliest feasible starting time is $t = 0$. But since θ is a

feasible point, we must also have $T_{i_1} = 0$ because otherwise $T_i > 0$ for all $i \in \mathcal{N}$ and constraint (13) is violated. Now assume that the jobs i_1, \dots, i_{l-1} have the right property and consider the job i_l . Let t_{i_l} be the first feasible time given that the jobs i_1, \dots, i_{l-1} have already been scheduled. If $t_{i_l} < T_{i_l}$ then θ violates the constraint (13) and is infeasible. This is hence a contradiction. If $t_{i_l} > T_{i_l}$ then the constraint (12) is violated and again θ is infeasible. Hence $t_{i_l} = T_{i_l}$. This holds for any $l \in \{2, 3, \dots, n\}$, so the entire sequence has the desired property.

Conversely, since the partitioning procedure never exceeds the available resources and each job is scheduled as soon as the resources allow, then every maximum depth region corresponds to a feasible schedule. ■

We have now established a valid partitioning procedure. The next step is to develop an efficient procedure to randomly sample a region.

5.2. Random sampling

Recall that the only constraint on a valid random sampling procedure is that each schedule should have a positive probability of being selected. When a region $\sigma \in \Sigma$ is sampled then $d(\sigma)$ jobs have already been assigned a fixed amount of resources and a starting time. For each of the $n - d(\sigma)$ jobs that have not been fixed the sampling procedure involves selecting cells, jobs, and resources. Sampling can always be done in a generic fashion, for example uniformly, but in practice it may be better to attempt to bias the sampling distribution towards good solutions. Since jobs selected to be fixed early tend to be scheduled earlier, and it is desirable to balance the workload, it is intuitive that higher priority should be given to selecting jobs in cells that have much work waiting. Also, it has some intuitive appeal to select with high probability resource levels that enable immediate scheduling of a job. This motivates the versatile sampling algorithm given below. The intuition behind this algorithm is that since the objective is to minimize the makespan it may be best to finish quickly as much work as possible in the cells that have the most work waiting. Hence priority is given to the most heavily loaded cells and long jobs in those cells.

Assume that a region $\sigma \in \Sigma$ is being sampled. Then the starting time and number of allocated resources has already been determined for the first $d(\sigma)$ jobs. The sampling algorithm fixes the remaining $n - d(\sigma)$ jobs iteratively and generates a single sample schedule $\hat{\theta} = (\hat{\mathbf{R}}, \hat{\mathbf{T}})$.

Sampling Algorithm

Step 0. Initialize $\mathcal{N}_j^U = \mathcal{N}_j^U(\sigma)$, $j \in \mathcal{M}$.

Repeat until $\mathcal{N}_j^U = \emptyset$ for all $j \in \mathcal{M}$.

Step 1. Determine the cell \hat{J} from which the next job is selected. Higher priority is given to cells that

have much work waiting to be processed, so \hat{J} is a random variable with

$$P[\hat{J} = j] = \begin{cases} \beta_1 \times \frac{\sum_{i \in \mathcal{N}_j^U} \hat{p}_{i1}}{\sum_{k \in \mathcal{M}} \sum_{i \in \mathcal{N}_k^U} \hat{p}_{i1}} + (1 - \beta_1) \\ \quad \times \frac{1}{\sum_{k \in \mathcal{M}} \chi_{\{i: \mathcal{N}_i^U \neq \emptyset\}}(k)}, & \mathcal{N}_j^U \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases} \quad (19)$$

Here β_1 is a constant to be determined.

Step 2. Given an outcome $\hat{j} \in \mathcal{M}$ select a job \hat{I} within this cell. Higher priority is given to long jobs, so \hat{I} is a random variable with

$$P[\hat{I} = i] = \begin{cases} \beta_2 \times \frac{\hat{p}_{i1}}{\sum_{k \in \mathcal{N}_{\hat{j}}^U} \hat{p}_{k1}} + (1 - \beta_2) \frac{1}{|\mathcal{N}_{\hat{j}}^U|}, \\ \quad i \in \mathcal{N}_{\hat{j}}^U, \\ 0 & \text{otherwise.} \end{cases} \quad (20)$$

Here β_2 is a constant to be determined.

Step 3. Given a selected job $\hat{i} \in \mathcal{N}$ select the number of resources \hat{R} given to this job. Higher priority is given to resource levels that allow for immediate scheduling, that is, resource levels that are less than or equal to $R_{\hat{j}}^A$, the available resources at the the earliest feasible time $t_{\hat{j}}^0$ in the cell $\hat{j} \in \mathcal{M}$. Then \hat{R} is a random variable with

$$P[\hat{R}_i = r] = \begin{cases} \beta_3 \frac{\beta_4}{R_{\hat{j}}^A} + (1 - \beta_3) \times \frac{1}{R}, & r \leq R_{\hat{j}}^A, \\ \beta_3 \frac{1 - \beta_4}{R - R_{\hat{j}}^A} + (1 - \beta_3) \times \frac{1}{R}, & r > R_{\hat{j}}^A, \end{cases} \quad (21)$$

Here β_3 and β_4 are constants to be determined. A weight β_4 is given to jobs that can be scheduled immediately and as before this biased sampling is weighted with uniform sampling.

Step 4. Schedule job $\hat{i} \in \mathcal{N}_{\hat{j}}$ using \hat{r} resources at the first feasible time $\hat{T}_{\hat{i}}$.

Step 5. If $\mathcal{N}_j^U = \emptyset$ for all $j \in \mathcal{M}$ then STOP. Otherwise go back to Step 1.

If $\beta_1 = \beta_2 = \beta_3 = 0$ then this is a uniform sampling procedure. If these constants are positive priority is given to cells that have many jobs waiting, long jobs, and resource allocations that enable immediate scheduling of the selected job. In particular, we refer to $\beta_1 = \beta_2 = \beta_3 = 1$ as weighted sampling. If $\beta_3 > 0$ then β_4 determines how much priority is given to resource levels that allow for immediate scheduling of the selected job.

Recall that the only condition on a valid random sampling scheme is that each schedule in the region must be selected with a positive probability. This is easily

verified for the above sampling algorithm and we state the following theorem without a proof.

Theorem 3. Let $\hat{\theta}$ be a random schedule with distribution that is defined by applying the sampling algorithm to a region $\sigma \in \Sigma$. For any $\beta_1, \beta_2, \beta_3 \in [0, 1]$, $\beta_4 \in (0, 1)$, and $\theta \in \sigma$, we have that $P[\hat{\theta} = \theta] > 0$.

5.3. Estimating the promising index

As for the generic implementation in Section 3.2 the estimated promising index may be taken as a simple summary statistic of the sampling information, that is, the smallest makespan from the sample points in each region. We let $\mathcal{D}(\sigma)$ denote the set of schedules sampled from region $\sigma \in \Sigma$ so this estimate is

$$\hat{I}(\sigma) = \min_{\theta \in \mathcal{D}(\sigma)} C(\theta).$$

However, if we have at our disposal an improvement heuristic $H_\sigma : \Theta \rightarrow \Theta$ that transforms an initial schedule $\theta \in \sigma$ into an improved schedule $\theta^* \in \sigma$, this can be incorporated into the promising index function as follows:

$$\hat{I}(\sigma) = \min_{\theta \in \mathcal{D}(\sigma)} C(H_\sigma(\theta)). \quad (22)$$

The improvement heuristic can, for example, be the efficient heuristic proposed in Daniels *et al.* [11]. However, the efficiency of the NP algorithm also depends on the random sampling algorithm, the structure imposed by the partitioning, and repeated use of the improvement heuristic. It may therefore be preferable to have a simpler heuristic that can be called repeatedly with relatively little computational cost. For example, we can seek simple changes to improve the resource allocation. Such improvements should be made by allocating more resources to critical jobs, that is, jobs which are such that a delay in the starting time of the job causes an increase in the makespan of the cell. This is the basic idea behind the improvement algorithm below.

Assume that a sample point $\theta = \{(R_i, T_i), \dots, (R_n, T_n)\}$ has been generated from $\sigma \in \Sigma$ using the random sampling procedure. Let \mathcal{J} denote the set of jobs that have been modified.

Improvement Algorithm

Step 0. Initialize $\mathcal{J} = \emptyset$.

Repeat until $\mathcal{N}_j^U(\sigma) \setminus \mathcal{J} = \emptyset$ for all $j \in \mathcal{M}$.

Step 1. Select a cell $\hat{j} \in \mathcal{M}$ such that $\mathcal{N}_{\hat{j}}^U(\sigma) \setminus \mathcal{J} \neq \emptyset$.

Step 2. For each $i \in \mathcal{N}_{\hat{j}}^U(\sigma)$ let $\tilde{\theta}_i = \{(R_i, T_i), \dots, (R_i, T_i + 1), \dots, (R_n, T_n)\}$, and calculate the set of critical jobs

$$\mathcal{C} = \left\{ i \in \mathcal{N}_{\hat{j}}^U(\sigma) : C_j(\tilde{\theta}_i) - C_j(\theta) > 0 \right\}. \quad (23)$$

If $\mathcal{C} = \emptyset$ let $\mathcal{J} = \mathcal{J} \cup \mathcal{N}_{\hat{j}}^U(\sigma)$ and go back to Step 1 above. Otherwise select a job $\hat{i} \in \mathcal{C}$, let $\mathcal{J} = \mathcal{J} \cup \{\hat{i}\}$, and continue to Step 3 below.

Step 3. If there are additional resources available during the time at which job \hat{i} is being processed, that is, if $\tilde{R} = R - \sum_{i=1}^n R_i \times \chi_{\{T_i, \dots, T_i + \hat{p}_{R_i}\}}(T_i) > 0$, then there is a possibility of improvement. If such a possibility exists continue to Step 4. Otherwise, if $\mathcal{C} \cap (\mathcal{N} \setminus \mathcal{J}) \neq \emptyset$, go back to Step 2. Otherwise go back to Step 1.

Step 4. If increasing the resources to job \tilde{i} decreases the makespan of the cell then give it all available extra resources. That is, consider

$$\tilde{\theta} = \{(R_{i_1}, T_{i_1}), \dots, (R_{\tilde{i}} + \tilde{R}, T_{\tilde{i}}), \dots, (R_{i_n}, T_{i_n})\},$$

and if $C_j(\tilde{\theta}) < C_j(\theta)$ then let $\theta = \tilde{\theta}$.

Step 5. If $\mathcal{C} \cap (\mathcal{N} \setminus \mathcal{J}) \neq \emptyset$, go to Step 2. Otherwise go to Step 1.

For this to be a valid heuristic to use to estimate the promising index we must check that when it is applied to a schedule in a given region $\sigma \in \Sigma$ then the improved schedule is also in the region, and in particular that the estimated promising index agrees with the original performance measure on singleton regions.

Theorem 4. Let $H_\sigma : \Theta \rightarrow \Theta$ denote the improvement algorithm described above.

(a) For all $\sigma \in \Sigma$,

$$H_\sigma(\sigma) \subseteq \sigma. \quad (24)$$

In particular if $\sigma = \{\theta\}$ is a singleton region then $H_\sigma(\theta) = \theta$ and

$$\hat{I}(\sigma) = C(\theta). \quad (25)$$

(b) For any $\theta \in \Theta$ such that $H_\sigma(\theta) \neq \theta$,

$$C(H_\sigma(\theta)) \leq C(\theta). \quad (26)$$

Proof. (a) Equation (24) follows directly from the observation that when applied to a region $\sigma \in \Sigma$ the improvement algorithm only makes changes to the jobs in $\mathcal{N}^U(\sigma)$. When $\sigma = \{\theta\}$ is a singleton region then $H_\sigma(\theta) = \theta$ and by Equation (22) we have

$$\hat{I}(\sigma) = \min_{\theta \in \mathcal{D}(\sigma)} C(H(\theta)) = C(\theta),$$

which proves Equation (25).

(b) Equation (26) holds because by Step 4 of the algorithm, changes in the schedule are only made if it decreases the makespan. ■

The sampling algorithm given in the last subsection, and the improvement heuristic given in above, are both intuitive and take advantage of the special structure of the PMFRS problem to speed convergence. However, it is important to note that these are not the only sampling and improvement schemes that can be used in the NP method. Any efficient methods can be used as long as Theorem 3 and Theorem 4(a) hold.

6. Numerical example

We now illustrate the NP method for the reformulated PMFRS problem with a few numerical examples. The implementation of the NP algorithm is as follows. The sampling algorithm is chosen to be uniform ($\beta_1 = \beta_2 = \beta_3 = 0$) in the subregions, and weighted ($\beta_1 = \beta_2 = \beta_3 = 1$) in the surrounding region with weight $\beta_4 = 1/2$. In each subregion one sample schedule is constructed, and 10 in the surrounding region. This increased computational effort in the surrounding region helps to enable the algorithm to backtrack when needed. In all regions the promising index is estimated by applying the improvement heuristic to the sample schedule(s). The processing times are generated in the following manner. For a job $i \in \mathcal{N}$, we let \hat{p}_{i1} be generated from a uniform random variable $\mathcal{U}(10, 50)$, and the remaining processing times calculated according to,

$$\hat{p}_{ir} = \left(1 - \alpha \left(1 - \frac{1}{k}\right)\right) \hat{p}_{i1},$$

where α is a constant that determines the speedup achieved when additional resources are available.

We start by considering how many iterations of the algorithm are required for obtaining near optimal schedules. To that end we look at problem with settings $n \in \{10, 20\}$, $R = 4$, $m \in \{3, 4, 5\}$, and $\alpha \in \{0.2, 0.5, 0.8\}$, for 100, 500, and 1000 iterations. The algorithm is repeated 100 times for each setting, so the averages are based on 100 independent replications. The results can be found in Table 1, that shows both the average and best makespan found for these settings. From the data we see that substantial improvements in average makespan are obtain by increasing the number of iterations from 100 to 500, but beyond that relatively little improvements are made for these examples. Furthermore, we note that for several setting the best makespan across all the replication is found when only 100 iterations are used, indicating that instead of using very long runs of the algorithm it may be more beneficial to use independent replications. Hence for the following numerical experiments we choose to use a moderate amount of 500 iterations.

Since optimal schedules for the PMFRS problem cannot be found efficiently, and the problems here are too large for complete enumeration, we evaluate the quality of schedules found by the NP algorithm by comparing the makespan of those schedules with the makespan of the optimal schedules for the static problem. Recall that for this problem only the resource allocation is of consequence, and an optimal schedule can be obtained efficiently. Since the makespan of the schedules obtained by the NP algorithm are also upper bounds for the optimal makespan for the PMFRS problem, this comparison also provides a lower bound on the performance improvement that may be achieved by making static resources flexible.

Table 1. Performance of the NP algorithm for variable number of iterations

<i>Problem settings</i>				<i>100 iterations</i>		<i>500 iterations</i>		<i>1000 iterations</i>	
<i>n</i>	<i>R</i>	α	<i>m</i>	<i>Average</i>	<i>Best</i>	<i>Average</i>	<i>Best</i>	<i>Average</i>	<i>Best</i>
10	4	0.2	3	101.8	99	100.8	98	100.6	98
			4	86.3	84	86.0	82	86.3	84
			5	79.2	75	78.2	75	77.5	74
		0.5	3	92.9	87	91.3	87	91.4	88
			4	81.2	76	80.4	76	80.1	76
			5	76.4	70	75.6	70	74.9	70
		0.8	3	79.7	75	78.4	75	77.8	74
			4	74.1	71	72.9	71	72.5	70
			5	72.0	69	71.1	69	70.9	68
20	4	0.2	3	193.0	184	193.0	186	191.3	181
			4	172.3	161	170.6	162	170.9	158
			5	162.0	137	160.7	133	159.9	154
		0.5	3	179.9	167	178.7	169	178.9	166
			4	163.1	148	162.6	155	161.1	152
			5	154.9	148	153.3	145	152.7	137
		0.8	3	160.3	147	159.6	149	158.9	152
			4	150.5	146	149.1	143	149.1	145
			5	144.1	138	142.9	138	142.6	138

Table 2. Performance of the NP algorithm for $n = 10$ after 500 iterations

<i>Problem settings</i>				<i>Makespan found by NP</i>			<i>Static problem</i>		<i>Average CPU time</i>
<i>n</i>	<i>R</i>	α	<i>m</i>	<i>Average</i>	<i>Min</i>	<i>SE</i>	<i>Makespan</i>	<i>% over (%)</i>	
10	4	0.2	3	100.8	98	1.8	100	2.0	22.6
			4	86.0	82	2.6	91	11.0	16.5
			5	78.2	75	7.0	†	19.2	
		0.5	3	91.3	87	2.7	100	14.9	22.5
			4	80.4	76	6.2	91	19.7	21.3
			5	75.6	70	6.0	†	23.6	
		0.8	3	78.4	75	3.3	100	33.3	12.4
			4	72.9	71	3.6	91	28.2	10.8
			5	71.1	69	1.9	†	15.6	
	5	0.2	3	98.6	96	1.8	99	3.1	14.0
			4	82.1	79	5.4	80	1.3	35.3
			5	74.5	70	3.9	79	12.9	23.9
		0.5	3	84.9	80	5.2	84	5.0	24.1
			4	72.9	68	7.9	71	4.4	21.1
			5	68.7	65	5.3	79	21.5	15.0
		0.8	3	68.6	64	3.1	67	4.7	15.9
			4	63.4	61	1.7	71	16.4	17.8
			5	61.3	59	3.0	79	33.9	25.0
6	0.2	3	96.5	95	1.1	96	1.1	32.2	
		4	79.4	77	4.0	77	0.0	18.8	
		5	71.7	67	4.2	70	4.5	20.3	
	0.5	3	79.6	74	4.3	75	1.4	12.7	
		4	67.6	64	5.4	68	6.3	22.2	
		5	63.1	58	6.2	63	8.6	25.2	
	0.8	3	61.7	58	2.9	60	3.4	34.4	
		4	57.3	52	2.7	63	21.2	18.2	
		5	53.6	49	3.0	63	28.6	23.5	

† Problem setting infeasible for static problem

We run the NP method using the above settings for problems with $n \in \{10, 15\}$ jobs, $m \in \{3, 4, 5\}$ cells, $R \in \{4, 5, 6\}$ flexible resources, and a speedup factor of $\alpha \in \{0.2, 0.5, 0.8\}$. For each of these settings we use 100 replications. The results are reported in Tables 2 and 3, and show the average and best makespan across the replications, as well as the standard error. The average CPU time in seconds is also reported. Since a fixed number of iterations is used, this does not vary much from one setting to another. Finally, as stated above, we report the makespan of the optimal static schedule for comparison, as well as the percentage improvement of the best PMFRS schedule found by the NP algorithm. The schedules found by the NP algorithm tend to have a much better performance than the static schedules, which indicates that considerable performance benefits may be obtained through the use of flexible resources. As is to be expected, the performance improvements increase with the speedup factor. Furthermore, the performance improvements also increase with the number of cells (m), indicating that flexible resources are most beneficial for systems where the speedup factor is large and there is a large number of parallel cells.

7. Conclusions

We have addressed the problem of simultaneously obtaining a resource allocation and sequencing jobs in a cellular manufacturing system with parallel cells and flexible resources. We developed a new method for solving this Parallel-Machine Flexible-Resource Scheduling (PMFRS) problem, called the Nested Partitions (NP) method. This method is an adaptive sampling method that combines local and global search and has been shown to converge in finite time for combinatorial problems [15]. We showed how the NP method can be implemented for the PMFRS problem and how the problem can be reformulated so that the NP method may take advantage of the special structure of the problem in both the global and local search. In doing that we developed a new sampling algorithm that biases the sampling towards good schedules, and a simple resource allocation improvement heuristic.

Our numerical results indicate that high quality schedules may be obtained using the NP method, and that substantial benefits may be derived from using flexible resources rather than static resources. The NP

Table 3. Performance of the NP algorithm for $n = 15$ after 500 iterations

Problem settings				Makespan found by NP			Static problem		Average CPU time		
n	R	α	m	Average	Min	SE	Makespan	% over (%)			
15	4	0.2	3	148.6	139	12.7	139	0.0	23.7		
			4	130.0	125	27.3	131	4.8	35.8		
			5	123.7	109	60.6	†		21.8		
		0.5	3	137.4	127	15.4	127	0.0	28.5		
			4	124.7	115	18.6	131	13.9	33.3		
			5	118.6	109	29.4	†		54.8		
		0.8	3	122.7	114	7.1	127	11.4	33.3		
			4	115.6	110	6.9	131	19.1	20.8		
			5	111.6	107	9.8	†		26.8		
	5	0.2	3	3	143.2	134	10.8	134	0.0	25.8	
				4	123.4	115	10.8	115	0.0	32.0	
				5	111.5	101	49.0	106	5.0	44.5	
			0.5	3	126.7	117	18.5	126	7.7	31.7	
				4	114.1	103	20.3	114	10.7	23.2	
				5	105.0	100	19.2	106	6.0	44.4	
		0.8	3	108.0	102	6.8	126	23.5	41.9		
			4	101.6	96	7.2	114	18.8	31.6		
			5	96.4	89	6.8	106	19.1	20.8		
		6	0.2	3	3	139.8	132	11.7	131	1.1	32.7
					4	119.8	112	12.8	114	1.8	39.2
					5	104.6	98	21.5	101	4.5	46.6
				0.5	3	120.8	108	17.9	116	7.4	22.8
					4	107.9	98	17.7	101	3.1	32.3
					5	98.2	90	14.3	101	12.2	61.9
0.8	3		98.0	91	6.0	93	3.3	54.4			
	4		91.1	86	5.8	101	17.4	25.0			
	5		86.5	81	7.0	101	24.7	49.0			

† Problem setting infeasible for static problem

method is expected to be particularly useful for very large problems, and can generate high quality schedules quickly. Future research will focus on further empirical testing of the method, incorporating other heuristics to speed convergence, and developing stopping rules for the method. Finally, we will also consider applying the NP method to the stochastic PMFRS problem where the processing times are subject to uncertainty.

Acknowledgments

This research was supported in part by the National Science Foundation under grant DMI-9713647.

References

- [1] Cheng, T.C.E. and Sin, C.C.S. (1990) A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, **47**, 271–292.
- [2] Pinedo, M. (1995) *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs, NJ.
- [3] Adiri, I. and Yehudai, Z. (1987) Scheduling on machines with variable service rates. *Computers and Operations Research*, **14**, 289–297.
- [4] Trick, M.A. (1994) Scheduling multiple variable-speed machines. *Operations Research*, **42**, 234–248.
- [5] Karabati, S. and Kouvelis, P. (1997) Flow-line scheduling problem with controllable processing times. *IIE Transactions*, **29**, 1–15.
- [6] Frenk, H., Labbé, M., van Vliet, M. and Zhang, S. (1994) Improved algorithms for machine allocation in manufacturing systems. *Operations Research*, **42**, 523–530.
- [7] Boxma, O.J., Rinnooy Kan, A.H.G. and van Vliet, M. (1990) Machine allocation problems in manufacturing networks. *European Journal of Operational Research*, **45**, 47–54.
- [8] So, K.C. (1990) Some heuristics for scheduling jobs on parallel machines with setups. *Management Science*, **36**, 467–475.
- [9] Daniels, R.L. and Mazzola, J.B. (1994) Flow shop scheduling with resource flexibility. *Operations Research*, **42**, 504–522.
- [10] Karabati, S., Kouvelis, P. and Yu, G. (1995) The discrete resource allocation problem in flow lines. *Management Science*, **41**, 1417–1430.
- [11] Daniels, R.L., Hoopes, B.J. and Mazzola, J.B. (1996) Scheduling parallel manufacturing cells with resource flexibility. *Management Science*, **42**, 1260–1276.
- [12] Vickson, R.G. (1980) Choosing the job sequence and processing times to minimize processing plus flow cost on a single machine. *Operations Research*, **28**, 1115–1167.
- [13] van Wassenhove, L.H. and Baker, K.R. (1982) A bicriterion approach to time cost trade-offs in sequencing. *European Journal of Operational Research*, **11**, 48–54.
- [14] Daniels, R.L. and Sarin, R.K. (1989) Single machine scheduling with controllable processing times and number of jobs tardy. *Operations Research*, **37**, 981–984.
- [15] Shi, L. and Ólafsson, S. (1996) Nested partitions method for global optimization. *Operations Research*, (in press).

Biographies

Sigurður Ólafsson is an Assistant Professor in the Department of Industrial and Manufacturing Systems Engineering at Iowa State University. He received a B.S. in Mathematics from the University of Iceland in 1995, an M.S. and Ph.D. in Industrial Engineering from the University of Wisconsin - Madison in 1996, and 1998, respectively. His research interests include scheduling, stochastic optimization, and simulation. He is a member of IIE, INFORMS, and ASEE.

Leyuan Shi is an Assistant Professor in the Department of Industrial Engineering at the University of Wisconsin-Madison. She holds a B.S. degree in Mathematics from Nanjing Normal University, China (1982), an M.S. degree in Applied Mathematics from Tsinghua University, China (1985), and an M.S. and a Ph.D. degrees in Applied Mathematics from Harvard University (1990,1992). Her research interests include modeling, analysis, and optimization of discrete event systems, discrete-event simulation, and sensitivity analysis. She is a member of IIE, INFORMS, and IEEE.

This paper is part of the Special Issue on Manufacturing Logistics