Extensions of an Integrated Approach for Multi-Resource Shop Scheduling*

Stéphane Dauzère-Pérès[a†] Claire Pavageau[b]

[a] Department of Finance and Management Science
Norwegian School of Economics and Business Administration
Helleveien 30
N-5035 Bergen-Sandviken, Norway

[b] Department of Automatic Control and Production Engineering
Ecole des Mines de Nantes
La Chantrerie, BP 20722
F-44307 Nantes Cedex 03, France

E-mail: {Stephane.Dauzere-Peres, Claire.Pavageau}@emn.fr

June 30, 1999

Abstract

In a previous work, we proposed an integrated approach for a rather general shop scheduling problem, with multi-resource, flexibility and non-linear routings. In this paper, we want to overcome some of the limitations of the approach. In particular, an operation that needs several resources might not need all the resources during its entire processing time. Our first extension allows a resource to be released before the end of the operation. The second extension considers the fact that, for a given operation, one might have to prevent a set of incompatible resources to be chosen.

1 Introduction

This paper considers a rather general shop scheduling problem in which, as in many practical applications, an operation may need several resources to be processed, and each of these resources may be chosen in a given set. Moreover, the routing of the products in the shop-floor is not necessarily linear, i.e., an operation can have more than one predecessor and/or more than one successor on the routing. This is for instance the case in assembly or disassembly systems. To keep the problem general, we use the word *resource* instead of *machine*. A resource can perform only one operation at a time, and

*This research was partially supported by the CRITT Pays de la Loire Productique
†on leave from IRCyN/Ecole des Mines de Nantes

preemption is not allowed, i.e., once started, an operation cannot be interrupted. The duration of an operation depends on the resources on which it is assigned, i.e., resource-dependent processing times. The processing times are assumed to be integer and known in advance, and the set-up times between operations are either negligible or included in the processing times (sequence independent).

We are thus considering three types of extension to the standard scheduling problem: multi-resource, resource flexibility, and nonlinear routing.

- **Multi-resource:** an operation may need several resources at the same time to be performed.
- **Resource flexibility:** a resource may be selected in a given set.
- **Nonlinear routing:** in a standard job-shop, the routing is *linear*. Here, the number of predecessors and the number of successors of an operation in the routing may be larger than one.

The problem is thus to both determine an assignment and a sequence of the operations on the resources that minimize some criterion.

Being an extension of the standard job-shop scheduling problem, this problem is clearly $\mathcal{NP}$-hard. Few results are available in the literature, even for the simpler job-shop scheduling problem with resource flexibility, but with only one resource per operation and linear routings (Brucker and Schlie [3] were among the first to address this problem). The first papers mainly focus on the scheduling part ([3], [2], [7], [8]).

In all of the approaches mentioned above, the assignment of operations to resources and the sequencing of operations on the resources are treated separately. Either **directly**, i.e., assignment and sequencing are considered independently, or **indirectly** in a local search algorithm where reassignment and resequencing are two different types of transitions. The method in [4] is based on a new way of moving an operation so that reassignment and resequencing are **not** differentiated. In [5], the methodology was extended to the more general case where multi-resource and nonlinear routings are allowed.

Note that the scheduling problem considered in this paper, like any scheduling problem, can be seen as a special case of the Resource Constrained Project Scheduling Problem (RCPSP) ([6]) where multiple modes are allowed (see [1] for instance). A mode corresponds to a subset of resources on which an operation can be processed. The notion of mode implies that subsets of resources will not be allowed for the operation. In [5], this was not explicitely handled, since it was supposed that every combination of some sets of resources was possible. In this paper, we extend the approach to deal with forbidden combinations of ressources. It should be noted that general solving procedures for the RCPSP are usually not well suited to shop scheduling problems.

To tackle even more realistic real-worl problems, we need to relax two important assumptions behind the approach proposed in [5]. The first one is that an operation starts only when all its assigned necessary resources are available, and releases these resources simultaneously. The processing time of an operation corresponds to the maximum of the processing times on all the assigned resources. This is clearly now always the case in practice, where a resource (often human) might be released and start another operation before the operation is completed. Not considering this case might lead to a poor schedule where bottleneck resources are under-utilized. The second assumption, already discussed,

is that any possible grouping of necessary ressources, each one taken in a given set, is possible. Again, this might not be allowed or wanted in a practice, because some resources are too far from one another or any other qualitative reasons that the workshop manager might have.

Our problem is described, and the notation is introduced in Section 2. The integrated approach introduced in [4] and [5] is recalled in Section 3. Then, Section 4 shows how this approach can be extended to tackle different processing times on the necessary resources. Forbidden sets of necessary resources are dealt with in Section 5. Finally, areas of future research are discussed in Section 6.

2 Description of the problem

A set of operations ($\mathcal{O}$) has to be processed on a set of resources $\mathcal{R}$. As nonlinear routings are permitted, $\mathcal{PR}(i)$ and $\mathcal{FR}(i)$ respectively denote the set of predecessors and the set of successors of operation i on the routing. To be processed, an operation $i \in \mathcal{O}$ requires $m(i)$ resources simultaneously and $\mathcal{R}_i^k$ is the resource subset in which the k^{th} resource $(1 \leq k \leq m(i))$ must be selected. The $\mathcal{R}_i^k$ subsets are not necessarily disjoint, i.e., a resource may belong to several subsets.

Solving the assignment problem means determining a mapping a: $\mathcal{O} \times \mathbb{N} \rightarrow \mathcal{R}$, where $a(i,k)$ is the k^{th} resource assigned to i, $a(i,k) \in \mathcal{R}_i^k$. Moreover we note $\mathcal{M}_i^a$ the subset of resources operation i is assigned to:

$$\mathcal{M}_i^a = \{m \in \mathcal{R} \mid \forall k \leq m(i),\ a(i,k) = m\}$$

As the $\mathcal{R}_i^k$ subsets are not necessarily disjoint, one must take care of not assigning a resource twice to the same operation, i.e., $\forall\ i,k,k'\ (k \neq k')$, $a(i,k) \neq a(i,k')$.

Together with an assignment a, the sequence of operations on each resource has also to be determined. We note $ps(i,k)$ the predecessor of i on its k^{th} resource and $fs(i,k)$ its successor. Then, for a given assignment a, let $\mathcal{PS}(i) = [ps(i,1), ..., ps(i,m(i))]$ be the index set of predecessors of operation i in a feasible sequence, and let $\mathcal{FS}(i) = [fs(i,1), ..., fs(i,m(i))]$ be the index set of its successors. All the operations do not have to be distinct in $\mathcal{PS}(i)$ or in $\mathcal{FS}(i)$. Finally, $\mathcal{P}(i) = \mathcal{PR}(i) \cup \mathcal{PS}(i)$ denotes the set of all predecessors of i, and $\mathcal{F}(i) = \mathcal{FR}(i) \cup \mathcal{FS}(i)$ denotes the set of all its successors.

A large class of shop scheduling problems can be handled using this modeling. In [5], the possibility of having differents modes, like in the Resource Constrained Project Scheduling Problem (RCPSP) (see [1] for instance), is not considered. The advantages of our modeling is that our subsets of resources are very natural in shop scheduling, since they are often associated to actual pools of identical resources, and hence are defined for more than one operation. Moreover, less data are required if the cardinals of the subsets are large since, in the usual RCPSP modeling, a mode (i.e., a subset of resources) has to be created for every possible combination. However, the drawback is that a priori all groupings of ressources (one taken in each set) are allowed, and this is clearly not always the case in industry where some assignment may be forbidden. For instance, it might be preferable not to assign an operator on some of the machines for geographical reasons, or to associate two given operators on a task for personal reasons. In this paper, we will show how to extend the approach to deal with forbidden sets of necessary resources.

In the job-shop scheduling problem, the processing time of an operation is fixed. In the multiprocessor job-shop ([4]), the duration of an operation i depends on the resource l on which it is assigned. In the general case considered here, this duration also depends on the subset k, and is noted $p^l_{i,k}$ where $l \in \mathcal{R}^k_i$. Hence, for a given assignment a, $p^{a(i,k)}_{i,k}$ is the duration of operation i on its k^{th} resource. In [5], p_i was defined as the *total* processing time and such that $p_i = \max_{k \in [1,m(i)]} p^{a(i,k)}_{i,k}$. This supposed that all assigned ressources have to be ready before starting an operation, and that they will all be released simultaneously. Again, in actual workshops, the operation might be started on some resources before others are available and, more often, might be released as soon as the operation is completed on *that* resource. Again, we will extend the approach of [5] to manage these conditions. To do so, let us define $start(i,k)$ which is equal to 1 if operation i needs to start on its k^{th} necessary resource when all other necessary resources are ready, and 0 otherwise. Similarly, let $end(i,k)$ be equal to 1 if operation i is only released on its k^{th} necessary resource when all other necessary resources are already released, and 0 otherwise. We are making the assumption that, for any given operation i and any of its necessary resource k, either $start(i,k) = 1$ or $end(i,k) = 1$. This is not restrictive since if an operation can start and end on one of its necessary resource independently of all the other resources, then it can and will better be modeled as another operation.

3 The integrated approach

In [5], the classical disjunctive graph representation $G = (N, A, E)$ was extended to handle resource flexibility, multi-resource and nonlinear routings. N is the set of nodes in the graph (all the operations plus dummy start and finish operations 0 and $\star$). A is the set of conjunctive arcs between every two consecutive operations in a routing. E_l is the set of disjunctive arcs between pairs of operations that *may* be processed on l, and $E = \bigcup_l E_l$. Note that two operations may be assigned to common resources, and there will be as many disjunctive arcs between these operations as the number of common resources.

It is shown in [5] how to characterize a complete selection S which is (roughly) determined by choosing an assignment for each operation and a direction for each associated disjunctive arc, i.e., by replacing a disjunctive arc with a conjunctive arc. Let S_l be the subset of S associated to resource l. A sequence of operations on the resources is **feasible** if the complete selection induces an *acyclic* (with no cycle) directed graph (N, A, S). Every operation is then linked by a conjunctive arc to all the operations processed on the same resource. In a complete acyclic selection S, and because the transitivity property is satisfied ($(i,j), (j,k) \in S_r \Rightarrow (i,k) \in S_r$), all the redundant arcs can be deleted to obtain an acyclic graph where, in S, $m(i)$ (and only $m(i)$) arcs will end up at i, and $m(i)$ (and only $m(i)$) arcs will start from i. More details can be found in [5].

An important but somehow restrictive assumption made in [5] is that the resources are released simultaneously when the operation is finished, i.e., as shown before, the processing time of an operation is the maximum of the processing times on the selected resources.

Example 1 *Let us consider for instance an operation i that requires a machine and a human operator to be performed. The machine has to be Machine $M1$, but the operator has to be chosen between two Operators $O1$ and $O2$. The processing of operation i requires*

p1 time units on M1, and p2 time units on either O1 or O2. Hence, $p_i = \max(p1, p2)$. *This is illustrated on Figure 1.*

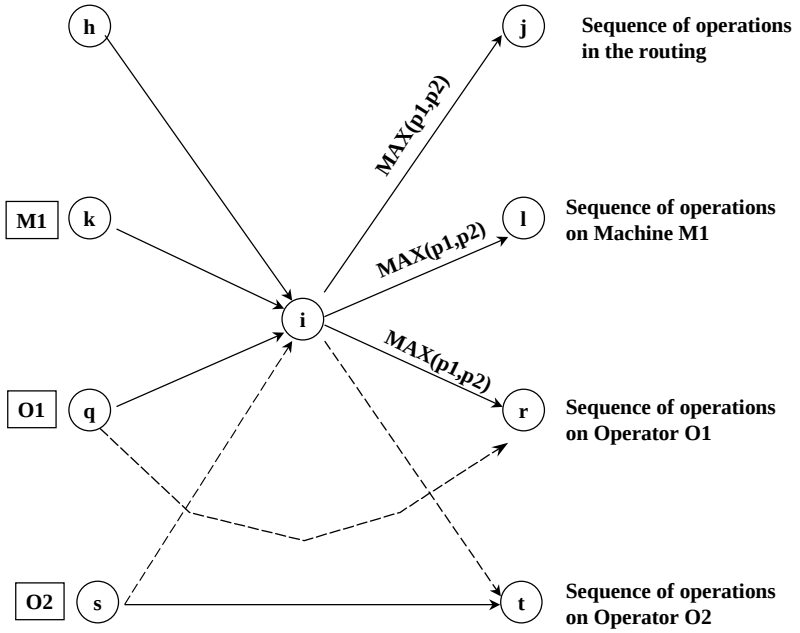

Figure 1: Example of flexibility and multi-resource

Operations h and j are the predecessor and successor of i in its routing, respectively. Operation i is sequenced between operations k and l on Machine M1, and between operations q and r on the selected Operator O1. Hence, operation i is not scheduled on Operator O2.

When a neighborhood is explored, a move consists in selecting an operation i on a resource l and inserting it between two operations s and t on a resource l', l' being equal or not to l. For this move to be valid, it should not induce a cycle in the resulting graph. In the previous example, operation i on Resource $O1$ is selected, and is reassigned on Resource $O2$, and is sequenced between operations s and t (see Figure 1).

To verify that no cycle is created when choosing a move, the following theorem is used.

Theorem 1 [5] *No cycle is created by moving operation i on its* k^{th} *resource between s* $(s \notin \mathcal{F}(i) - \{fs(i,k)\})$ *and t* $(t \notin \mathcal{P}(i) - \{ps(i,k)\})$, $(s,t) \in S_{l'}$ *and* $l' \in \mathcal{R}_i^k$, *if:*

1. $r_s < \min_{f \in \mathcal{F}(i) - \{fs(i,k)\}} (r_f + p_f)$
2. $r_t + p_t > \max_{p \in \mathcal{P}(i) - \{ps(i,k)\}} r_p$

Because the same idea will be used in the extensions presented in the remainder of this paper, it is important to sketch the proof of Theorem 1. It is based on the idea that, if a cycle is created after completing a move, then, in the graph before the move, there

was a path either between an operation after i (except $fs(i,k)$) and s, or between t and an operation after i (except $ps(i,k)$). The first situation is prevented by Conditions 1, and the second by Conditions 2 (see details in [5]). These conditions can be checked very quickly, and the neighborhood, although large, can be rapidly explored since moves do not have to be performed to check their feasibility. This theorem is actually coupled with another theorem that allows a move to be evaluated without making it.

The neighborhood structured induced by the conditions of Theorem 1, is connected, i.e., an optimal solution can be reached from any starting solution in a finite number of steps (see [5]). The proof is partly based on the fact that, for the classical job-shop scheduling problem, the well-known *pairwise interchange* move leads to a connected neighborhood structure (see [9]). This move uses the fact that exchanging the direction of a disjunctive arc on the critical path does not create a cycle. However, this is true only if, when an arc is on a critical path, there is not another path parallel to this arc. To be more precise, if the arc between operation i and j is on the critical path, then there is not another path between i and j since, by exchanging the direction of the arc, a cycle is created. We know that this cannot happen if all arcs leaving a node (i.e., an operation) have the same weight (i.e., in our case, same processing time on all the resources). This is true in our example since the processing time is $\max(p1, p2)$.

Note that Theorem 1 is still valid in a graph where all arcs leaving a given node do not always have the same weights. However, in this case, we cannot prove that our neighborhood structure is connected any more.

4 Differentiating processing times on necessary resources

Suppose now that the processing times on the various necessary resources *have* to be different. For instance, the operator needs only several minutes at the start of the operation to check if nothing is wrong on the machine, and the remainder of the operation can be performed only by the machine (in our example, $p2 \ll p1$). The operator is free as soon as his work is completed, and can perform other operations. Taking this case into account might be crucial if the operator is the bottleneck resource. This could be modeled by allowing different weights on the arcs leaving the node of the operation. The weight of the arc, between the node and the next node in the sequence of the operator, will be equal to the processing time of the operator. The resulting graph ensures that the next operation of the operator starts after the start time of the operation plus the processing time *of the operator*. However, as discussed in Section 3, connectivity of our neighborhood structure is not guaranteed any more.

Hence, we would like to model the fact that a resource might be released before the end of the operation, while ensuring that all arcs leaving a node have the same weight. This is done by duplicating the nodes of the graph associated to operations that need several resources, so that a node now is coupled with only **one** necessary resource of **one** operation. This new representation also allows different start times on each of the necessary resources. An operation i is related to $m(i)$ nodes denoted $i_1, ..., i_{m(i)}$. Recall that $start(i,k)$ is equal to 1 if operation i starts on its k^{th} necessary resource when all other necessary resources are ready, and 0 otherwise, and that $end(i,k)$ is equal to 1 if operation i is not released on its k^{th} necessary resource before all other necessary

resources are released, and 0 otherwise. Because start times of an operation might differ between necessary resources, $r_{i,k}$ will denote the start time of operation i on its k^{th} necessary resource. Moreover, let us define $pk(i,k)$ (resp. $fk(i,k)$) the number of the necessary resource of $ps(i,k)$ (resp. $fs(i,k)$) which is before (resp. after) the k^{th} necessary resource of i. The definitions of $\mathcal{PS}(i)$ and $\mathcal{FS}(i)$ are updated accordingly. $\mathcal{PS}(i)$ (resp. $\mathcal{FS}(i)$) now contains the pair $(ps(i,k), pk(i,k))$ (resp. $(fs(i,k), fk(i,k))$) for every $k \in [1, m(i)]$, i.e., $\mathcal{PS}(i) = \{(ps(i,1), pk(i,1)), ..., (ps(i,m(i)), pk(i,m(i)))\}$ and $\mathcal{FS}(i) = \{(fs(i,1), fk(i,1)), ..., (fs(i,m(i)), fk(i,m(i)))\}$. $\mathcal{PS}(i,k)$ and $\mathcal{FS}(i,k)$ correspond to the pair $(ps(i,k), pk(i,k))$ and $(fs(i,k), fk(i,k))$ respectively.

For every two successive operations i and j in the routing ($i \in \mathcal{PR}(j)$ and $j \in \mathcal{FR}(i)$), there is an arc from every node associated to i $(i_1, ..., i_{m(i)})$ to every node associated to j $(j_1, ..., j_{m(j)})$. Let us now consider two consecutive operations i and j in the sequence of a resource l, the $k1^{th}$ necessary resource of i ($k1 \in [1, m(i)]$) and the $k2^{th}$ necessary resource of j ($k2 \in [1, m(j)]$), i.e., $(i, k1) \in \mathcal{PS}(j)$ and $(j, k2) \in \mathcal{FS}(i)$. There is an arc between i_{k1} and j_{k2}, but also between i_{k1} and every node j_k of operation j ($k \neq k2$) such that $start(j,k) = 1$, since it means that j can only start on its k^{th} necessary resource when all necessary resources are available. Moreover, if $end(i,k1) = 1$, there is an arc from every node of i $(i_1, ..., i_{m(i)})$ to node j_{k2}, but also to every node j_k of operation j ($k \neq k2$) such that $start(j,k) = 1$, since the $k1^{th}$ necessary resource of i is only released when all other necessary resources of i are released.

More formally, an arc will be added from node i_{k1} to node j_{k2} if

- (1) $j \in \mathcal{FR}(i)$ (or equivalently $i \in \mathcal{PR}(j)$)
- or (2) $\exists (k3, k4) \in [1, m(i)] \times [1, m(j)]$ such that $(j, k4) \in \mathcal{FS}(i)$ and $(i, k3) \in \mathcal{PS}(j)$ (i.e., $(j, k4) = \mathcal{FS}(i, k3)$ and $(i, k3) = \mathcal{PS}(j, k4)$), and

$$\Big[(k1 = k3 \text{ and } k2 = k4)$$
$$\text{or } (k1 \neq k3 \text{ and } k2 = k4 \text{ and } end(i,k1) = 1)$$
$$\text{or } (k1 = k3 \text{ and } k2 \neq k4 \text{ and } start(j,k2) = 1)$$
$$\text{or } (k1 \neq k3 \text{ and } k2 \neq k4,\ end(i,k1) = 1 \text{ and } start(j,k2) = 1)\Big]$$

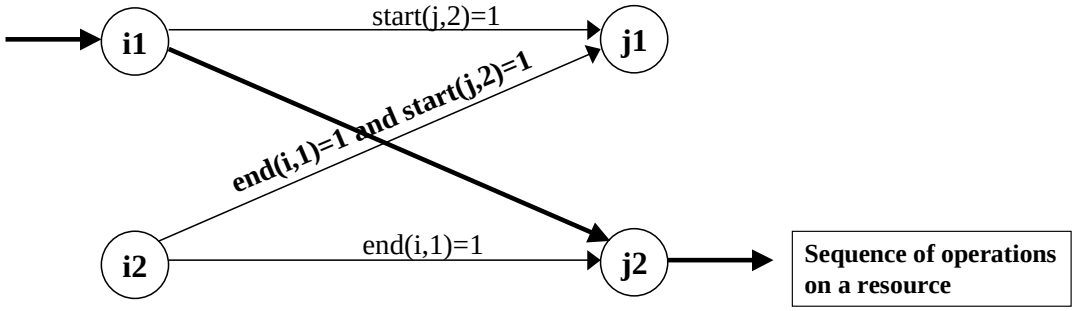

Figure 2: Example of new modeling between two operations

Figure 2 illustrates the conditions for creating arcs between nodes of two operations i and j with two necessary resources each. In this figure, we suppose that, in the sequence of a resource, operation i on its 1^{st} necessary resource is sequenced before operation j on its 2^{nd} necessary resource (node i_1 before node j_2).

Note that there might have several arcs between two nodes.

Remark 1 *Recall that we suppose (and it was not restrictive) that, for any given operation i and any of its necessary resource k, either $start(i,k) = 1$ or $end(i,k) = 1$. This assumption implies that there is always a path between the node associated to operation $ps(i,k)$ on its necessary resource $pk(i,k)$ and the node associated to operation $fs(i,k')$ on its necessary resource $fk(i,k')$, $\forall (k,k') \in [1,m(i)] \times [1,m(i)]$. If $start(i,k') = 1$, then there is an arc from node $ps(i,k)_{pk(i,k)}$ to node $i_{k'}$, which gives the path with the arc from $i_{k'}$ to $fs(i,k')_{fk(i,k')}$. If $end(i,k') = 1$, then there is an arc from node i_k to node $fs(i,k')_{fk(i,k')}$, which gives the path with the arc from $ps(i,k)_{p(i,k)}$ to i_k.*

Let us illustrate how start and end times are differentiated using the new modeling.

Example 2 *Using again Example 1, two nodes will be associated to operation i: Node i_1 for the first necessary resource, and Node i_2 for the second necessary resource. Node i_1 is assigned to Machine $M1$, and Node i_2 to Operator $O1$ (see Figure 3).*

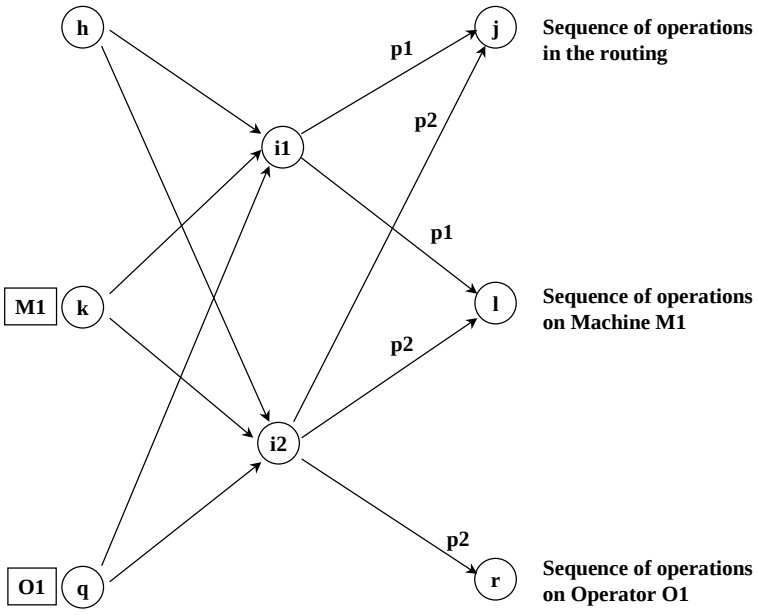

Figure 3: Differentiating end times on Ressources $M1$ and $O1$

Arcs (k,i_1) and (k,i_2) ensure that operation i will not start before Machine $M1$ is available, and Arcs (q,i_1) and (q,i_2) that operation i will not start before Operator $O1$ is available. Arcs (i_1,l) and (i_2,l) ensure that no operation starts on Machine $M1$ before operation i is completed, and Arc (i_2,r) that Operator $O1$ does not process another operation before completing operation i. This new modeling, by deleting Arc (i_1,r), does not

force Operator $O1$ to be busy on operation i as long as Machine $M1$. Moreover, one can check that all arcs leaving a node have the same weight. Note that adding Arc (i_1, r) will lead to a model that is equivalent to the previous one in Figure 1.

Let us show how we might allow for different start times on the necessary resources. For instance, suppose that, for operation i, Operator $O1$ can set-up the job before Machine $M1$ is available. In this case, operation i still requires both resources, but operator $O1$ can start operation i before Machine $M1$ becomes available (see Figure 4).

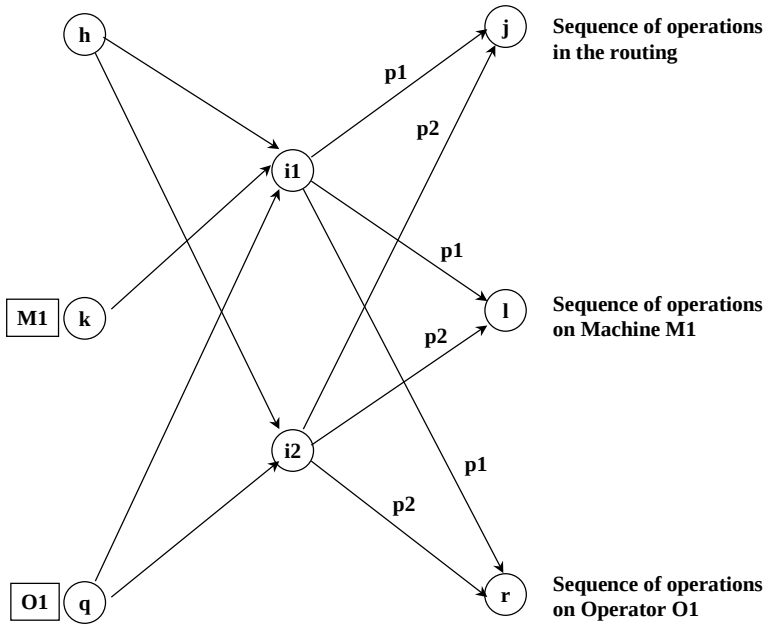

Figure 4: Differentiating start times on Ressources $M1$ and $O1$

Using the new modeling, it is possible, at a price of an an increase in the number of nodes and arcs, to differentiate the processing times of an opeartion on its various necessary resources, and to better schedule the resources.

The move in the neighborhood structure needs to be updated accordingly. Moving operation i on its k^{th} necessary ressource from resource l to resource l' (l' being equal to n or not) between operations s ($k1^{th}$ necessary resource) and t ($k2^{th}$ necessary resource) is performed by updating $\mathcal{PR}$ and $\mathcal{FR}$, and removing and adding arcs such that:

- $ps(i,k) = s$, $pk(i,k) = k1$, $fs(s,k1) = i$, and $fk(s,k1) = k$.
- $fs(i,k) = t$, $fk(i,k) = k2$, $ps(t,k2) = i$, and $pk(t,k2) = k$.
- $ps(fs(i,k), fk(i,k)) = ps(i,k)$ and $pk(fs(i,k), fk(i,k)) = pk(i,k)$.
- $fs(ps(i,k), pk(i,k)) = fs(i,k)$ and $fk(ps(i,k), pk(i,k)) = fk(i,k)$.

The first two items correspond to inserting operation i in the sequence of resource l', and the last two items to deleting operation i from the sequence of resource l.

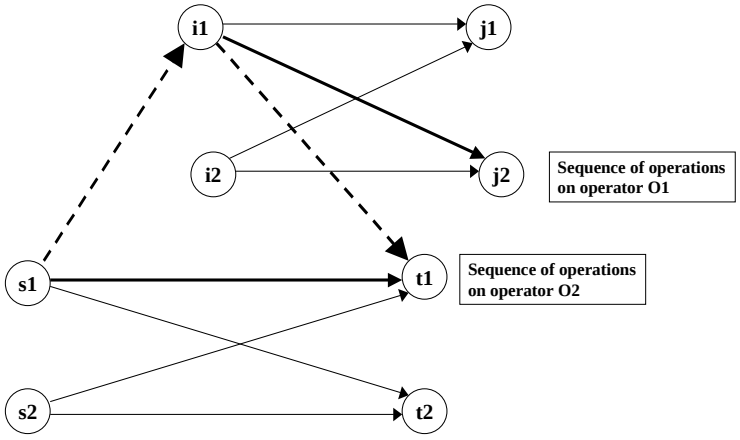

Figure 5: Before performing a move

Figures 5 and 6 illustrate how a move is performed when operation i is reassigned to operator $O2$. We suppose that operations s and t also have two necessary resources, that, before the move, node s_1 is sequenced just before node t_1 in the sequence of operator $O2$, and that $end(s,1) = end(i,1) = start(t,1) = 1$.

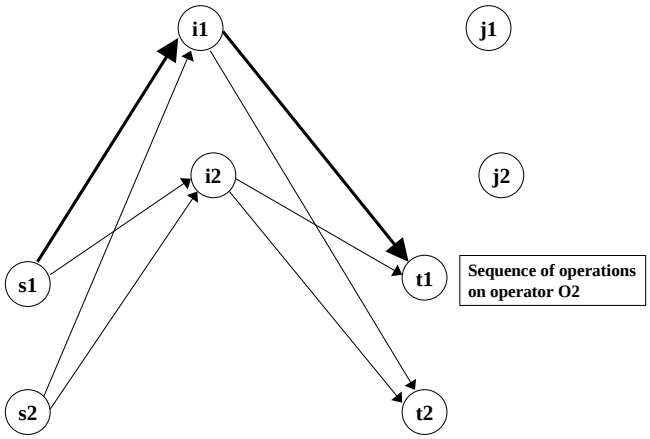

Figure 6: After performing a move

Theorem 1 needs to be changed accordingly.

Theorem 2 *No cycle is created by moving operation i on its k^{th} resource between the $k1^{th}$ necessary resource of s ($s \notin \mathcal{FR}(i)$ and $(s,k1) \notin \mathcal{FS}(i) - \{(fs(i,k), fk(i,k))\}$) and the $k2^{th}$ necessary resource of t ($t \notin \mathcal{PR}(i)$ and $t \notin \mathcal{PS}(i) - \{(ps(i,k), pk(i,k))\}$), $(s,t) \in S_{l'}$ and $l' \in \mathcal{R}_i^k$, if:*

1. $r_{s,k'} < \min_{f \in \mathcal{FR}(i)} \min_{k" \in [1,m(f)]} (r_{f,k"} + p_{f,k"}^{a(f,k")})$, $\forall k' \in [1, m(s)]$ *such that* $k' = k1$ *or* $end(s,k1) = 1$

2. $r_{t,k'} + p_{t,k'}^{a(t,k')} > \max_{p \in \mathcal{PR}(i)} \max_{k" \in [1,m(p)]} r_{p,k"}$, $\forall k' \in [1, m(t)]$ *such that* $k' = k2$ *or* $start(t,k') = 1$

3. $r_{s,k'} < \min_{(f,kf) \in \mathcal{FS}(i) - \{(fs(i,k), fk(i,k)\}} \min_{\substack{k" \in [1,m(f)]; \\ k" = kf \text{ or } start(f,k") = 1}} (r_{f,k"} + p_{f,k"}^{a(f,k")})$, $\forall k' \in [1, m(s)]$ *such that* $k' = k1$ *or* $end(s,k1) = 1$

4. $r_{t,k'} + p_{t,k'}^{a(t,k')} > \max_{(p,kp) \in \mathcal{PS}(i) - \{(ps(i,k), pk(i,k))\}} \max_{\substack{k" \in [1,m(p)]; \\ k" = kp \text{ or } end(p,kp) = 1}} r_{p,k"}$, $\forall k' \in [1, m(t)]$ *such that* $k' = k2$ *or* $start(t,k') = 1$

Proof: The proof uses the same principle that the one described in Section 3 for Theorem 1, i.e., prevent cycling by ensuring that, before the move, there was not a path between an operation moved after s and s and between t and an operation moved before t. The difference is that every operation now might be associated to several nodes, and that the existence of arcs between nodes depends also on the values of the parameters $start(i,k)$ and $end(i,k)$.
Hence, if a path is created between two nodes $s_{k'}$ and $f_{k"}$ in the graph obtained by moving operation i, then a cycle can only be created if there was a path, on the graph before the move, between $f_{k"}$ and $s_{k'}$. This is only true if the length of the longest path from the dummy start node 0 to node $s_{k'}$ is larger than the one from 0 to $f_{k"}$ plus its processing time, or equivalently the release date of $s_{k'}$ is larger than the one of $f_{k"}$ plus its processing time, i.e., $r_{s,k'} >= r_{f,k"} + p_{f,k"}^{a(f,k")}$. The opposite condition ($r_{s,k'} < r_{f,k"} + p_{f,k"}^{a(f,k")}$) ensures that this is not the case.
The problem is now to apply the previous condition only when an actual path is created between nodes $s_{k'}$ and $f_{k"}$. We know that, by definition, there is an arc from every node of operation i to every node of operations following i in the routing ($\in \mathcal{FR}(i)$). Hence, a path is created between every node of operation s linked to a node of operation i. This is the case for every node $s_{k'}$ such that $k' = k1$ or if $end(s,k1) = 1$. In this case, the cycle is prevented through Conditions 1. A similar analysis with operation t and nodes of operations preceding i in the routing ($\in \mathcal{PR}(i)$) leads to Conditions 2. The difference is that a node of operation i is linked to node $t_{k'}$ if $k' = k2$ or $start(t,k') = 1$.
The last two conditions use Remark 1. We know that there is a path between node $s_{k'}$ and node $f_{k"}$ if there is an arc from node $s_{k'}$ to a node of operation i (same conditions than for Conditions 1 and 2), and from a node of operation i to node $f_{k"}$. The latter is true if $\exists kf$, $(f,kf) \in \mathcal{FS}(i) - \{(fs(i,k), fk(i,k)\}$, such that $k" = kf$ or $start(f,k") = 1$. Cycles are then prevented using Conditions 3. Again, the same kind of analysis can be performed with nodes $t_{k'}$ and $p_{k"}$ to derive Conditions 4. The difference in this case is that there is an arc from node $p_{k"}$ to a node of i if $\exists kp$, $(p,kp) \in \mathcal{PS}(i) - \{ps(i,k), pk(i,k)\}$ such that $k" = kp$ or if $end(p,kp) = 1$. □

5 Forbidden sets of necessary resources

In some practical settings, it may very well happen that, although some flexibility is given on the choice of the necessary resources to select for a given operation, some combinations are not allowed. For instance, if you have two workshops that are able to perform the

same type of operations, there is no possibility of assigning an operator of the first workshop with a machine in the second workshop and vice-versa, although several operators and/or resources are available in each workshop, i.e., there is flexibility in each workshop (otherwise, each workshop could be considered as a single resource). In the move used in the previous approach, an operation is resequenced or reassigned on only *one* of its necessary resources at a time. Hence, it is clear that, if an operation has been assigned on a machine and an operator in a workshop, the neighborhood structure will not allow the operation to be assigned to resources in the other workshop. Again, the neighborhood structure loses its connectivity property.

The solution we propose is to move the operation on all its necessary resources at a time. In order to do that, and because we would still want to be able to check feasibility without actually making a move, we need to extend the conditions of Theorem 1. This is done in the following theorem.

Theorem 3 *No cycle is created by moving operation i on all its necessary resources between s_k ($s_k \notin \mathcal{FR}(i)$) and t_k ($t_k \notin \mathcal{PR}(i)$), $(s_k, t_k) \in S_{l'}$ and $l' \in \mathcal{R}_i^k$, $k = 1, ..., m(i)$, if:*

1. $r_{s_k} < \min_{f \in \mathcal{FR}(i)} (r_f + p_f) \qquad \forall k \in [1, m(i)]$

2. $r_{t_k} + p_{t_k} > \max_{p \in \mathcal{PR}(i)} r_p \qquad \forall k \in [1, m(i)]$

3. $r_{s_k} < r_{t_{k'}} + p_{t_{k'}} \qquad \forall (k, k') \in [1, m(i)] \times [1, m(i)]; k \neq k'$

Proof: The proof follows the ones of Theorems 1 or 2, except that all operations in the sets $\mathcal{PS}(i)$ and $\mathcal{FS}(i)$ are changed. Hence, the path that might create a cycle has to be checked for the new operations s_k and t_k, and not for the ones in $\mathcal{PS}(i)$ and $\mathcal{FS}(i)$. Conditions 1 and 2 correspond to Conditions 1 and 2 in Theorem 1 restricted to operations in $\mathcal{PR}(i)$ and $\mathcal{FR}(i)$, and for all the new operations s_k and t_k. Conditions 1 and 2 in Theorem 1 for sets $\mathcal{PS}(i)$ and $\mathcal{FS}(i)$ are replaced by Conditions 3 for all pairs of operations (s_k,$t_{k'}$, $k \neq k'$). □

The neighborhood for a given operation i induced by the conditions of Theorem 3 includes the one where i is only moved on one resource. This is because, as shown in the following proposition, when the operation is moved on only one of its necessary resources, the conditions of Theorem 3 are equivalent to the conditions of Theorem 1.

Proposition 1 *When operation i is only resequenced or reassigned on one of its necessary resources, conditions of Theorem 3 reduce to the conditions of Theorem 1.*

Proof: Suppose that operation i is moved on only one of its necessary resources, say $k1$ ($k1 \in [1, m(i)]$). Hence, s_k and t_k, except for $k = k1$, are the same predecessors and successors of i on its k^{th} resource, i.e., $s_k \in \mathcal{PS}(i)$ and $t_k \in \mathcal{FS}(i)$, $k \in [1, m(i)]$, $k \neq k1$. This implies that, for $k \neq k1$, Conditions 1 and 2 are satisfied (since s_k precedes i which in turn precedes f, $\forall f \in \mathcal{FR}(i)$, and t_k follows i which in turn follows p, $\forall p \in \mathcal{PR}(i)$). For the same reason, Condition 3 is also satisfied for $k \neq k1$ and $k' \neq k1$. Combining all Conditions 3 for $k = k1$ leads to $r_{s_{k1}} < \min_{t_k \in \mathcal{FS}(i) - \{fs(i,k1)\}} (r_{t_k} + p_{t_k})$

which, together with Condition 1, is equivalent to Condition 1 of Theorem 1 (recall that $\mathcal{F}(i) = \mathcal{FR}(i) \cup \mathcal{FS}(i)$). Similarly, combining all Conditions 3 for $k' = k1$ leads to $r_{t_{k1}} + p_{t_{k1}} > \max_{s_k \in \mathcal{PS}(i) - \{ps(i,k1)\}} r_{s_k}$ which, together with Condition 2, is equivalent to Condition 2 of Theorem 1 ($\mathcal{P}(i) = \mathcal{PR}(i) \cup \mathcal{PS}(i)$). □

Because of Proposition 1, we know that any solution that can be attained with the neighborhood defined by the conditions of Theorem 1 can also be attained with the neighborhood using the conditions of Theorem 3. However, the opposite is not true, since the conditions on moving an operation i on all its necessary resources $m(i)$ are less restrictive than to perform $m(i)$ different moves. It is then possible to find moves that will more quickly lead to a large decrease of the makespan. However, this is clearly done at the expense of a larger neighborhood. Hence, because the neighborhood is considerably enlarged, we propose to only perform this type of move only when necessary, i.e., only when an operation has forbidden sets of necessary resources.

6 Conclusion

We extended an integrated approach for multi-resource shop scheduling to two different realistic cases. We first model the fact that an operation might have different processing times on its necessary resources, and might then release resources at different times, by using several nodes to represent the operation in the graph. The original move is extended to take this change into consideration. A very interesting by-product of the new modeling is that it also allows different start times for the operation (not possible if there is only one node per operation). In the second case, we consider that some combinations of necessary resources might be forbidden. To avoid being stuck in poor local optima, we also extend the original move.

The latter case was dealt with by extending the conditions of Theorem 1. Because the move is rather different (an operation is moved on several of its necessary resources simultaneously), it might be worth finding new conditions, for preventing a cycle in the graph after a move, not directly related to the ones in Theorem 1.

It should also be very interesting to develop the approach for other criteria than the makespan. Although due dates can be rather easily taken into consideration if one wants to minimize the maximum lateness (can be transformed to an equivalent makespan problem), criteria such as the (weighted) sum of the lateness might be harder to handle.

References

[1] Bianco, L., Dell'Olmo, P., and Speranza, M.G. "Nonpreemptive scheduling of independent tasks with prespecified processor allocations", *Naval Research Logistics* 41, (1994) 959-971.

[2] Brandimarte, P., "Routing and scheduling in a flexible job shop by tabu search", *Annals of Operations Research* 41 (1993) 57-183.

[3] Brucker, P., and Schlie, R.,"Job-shop scheduling with multi-purpose machines", *Computing* 45 (1990) 369-375.

[4] Dauzère-Pérès, S., and Paulli, J., "An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search," *Annals of Operations Research* 70 (1997) 281-306, J.C. Baltzer A. G..

[5] Dauzère-Pérès S., Roux W., and Lasserre J.B., "Multi-Resource Shop Scheduling with Resource Flexibility," *European Journal of Operational Research* 107 (1998) 289-305.

[6] Herroelen, W., and Demeulemeester, E., "Recent advances in branch-and-bound procedures for resource-constraint project scheduling problems", in *Scheduling theory and its applications*, Chrétienne, P. *et al.* (eds.) (1995) 259-274, John Wiley & Sons, Chichester.

[7] Hurink, J., Jurisch, B., and Thole, M., "Tabu search fo the job-shop scheduling problem with multi-purpose machines", *OR Spektrum* 15 (1994) 205-215.

[8] Paulli, J.,"A hierarchical approach for the FMS scheduling problem", *European Journal of Operational Research* 86 (1995) 32-42.

[9] van Laarhoven, P.M., Aarts, E.H.L., and Lenstra, J.K., "Job shop scheduling by simulated annealing", *Operations Research* 40 (1992) 113-125.