Multiresource Shop Scheduling With Resource Flexibility and Blocking

Yazid Mati and Xiaolan Xie

***Abstract*—This paper proposes a general scheduling model that extends job-shop scheduling models to incorporate important features of real manufacturing systems. More precisely, each operation can be performed in different modes and requires a different set of resources depending on the mode. Further, we consider blocking constraints that requires to hold resources used for an operation till resources needed for the next operation of the same job are available. A shortest path approach extending the classical geometric approach is proposed for the two-job case. A greedy heuristic is then proposed to schedule N jobs by considering jobs sequentially, grouping scheduled jobs into a combined job and then scheduling it and the next unscheduled job using the shortest path approach. A metaheuristic is then used to identify effective job sequences. Extensive numerical experimentation proves the efficiency of our approach.**

***Note to Practitioners*—Many manufacturing systems such as the production of key components of aircrafts require large storage spaces and efficient material handling resources. This paper proposes a general scheduling model that extends job-shop scheduling models to incorporate important features of real manufacturing systems. More precisely, each operation can be performed in different modes and requires a different set of resources depending on the mode. Further, we consider blocking constraints that requires to hold resources used for an operation until resources needed for the next operation of the same job are available. An efficient approach combining a novel geometric approach for the two-job case and metaheuristics is proposed. Extensive numerical experimentation proves the efficiency of our approach.**

***Index Terms*—Blocking, deadlock, geometric approach, makespan, multiresource, resource flexibility, scheduling.**

I. Introduction

IN MANY manufacturing systems such as the production of key components of aircrafts, increasing the level of automation is crucial. Often products are of large size and heavy weight and are highly expensive. The storage space is highly limited and products often wait long time on machines after operations. Expensive transportation devices are needed for material handling. Each operation can often be performed on several machines in order to reach reasonable productivity.

Manuscript received March 23, 2010; accepted May 16, 2010. This paper was recommended for publication by Associate Editor S. G. Ponnambalam and Editor Y. Narahari upon evaluation of the reviewers' comments.

Y. Mati is with the College of Business and Economics, Al-Qassim University, Almelaida, Buraidah 15452, Kingdom of Saudi Arabia (e-mail: matie@qu.edu.sa).

X. Xie is with the Healthcare Engineering Department, Ecole des Mines de Saint-Etienne, 158, cours Fauriel 42023 Saint-Etienne cedex 2, France (e-mail: xie@emse.fr).

Digital Object Identifier 10.1109/TASE.2010.2052356

Such manufacturing systems are characterized by high-level automation. These systems use a variety of production resources such as NC machines that are able to perform several operations at a time, transportation resources such as robots and automated guided vehicles, automated storage devices with a very limited (or zero) buffer space, etc. Flexibility of production equipments and manufacturing processes is also introduced to face demand volatility.

Some existing scheduling models do incorporate some features encountered in modern production systems such as multiresource operations [10], and resource flexibility [9]. However, a strong limitation of most models is the assumption of unlimited buffer capacity between the workstations which allow the release of resources immediately after the completion of operations.

Buffer capacity constraints are usually taken into account through the so-called blocking constraints under which a job can leave a machine only if the next machine is available. The blocking constraints are often met in industrial production like aluminum, chemical and food processing industries [12]. It can also be found in other industrial settings like in computer architectures [1] in the context of packet switching communication networks. Blocking constraints are also important in the production of concrete blocks [11] and steel [24]. Another important application that has received an increasingly attention during recent years is railway scheduling [5] in which blocking actually leads to deadlock situations and routing flexibility of trains has been taken into account to increase the schedule efficiency. Note that existing scheduling models with blocking do not take into account resource flexibility and are mostly restricted to only machine requirement, i.e., single resource operations.

In this paper, we propose a general scheduling model that extends classical job-shop scheduling models to incorporate practical constraints by using three key features. It can also be considered as an extension of our previous scheduling model in [18] which does not take into account resource flexibility.

The first key feature of our model is the multiresource operations [10] for modeling production systems in which the realization of an operation may need more than one resource at a same time. For instance, an operation may need a machine, a tool and a human operator to be performed. The second feature is the resource flexibility [9] that allows an operation to be performed in different way with different resources. The third feature is the blocking constraint that requires to hold resources used for an operation till resources needed for the next operation of the same job are available. It extends the blocking constraints considered in the literature for flow shop systems [12]. Contrary to blocking

in flow-shops that only delays operations, blocking constraints in our case may lead to deadlock situations which could happen when automated material handling devices are used.

This paper is closely related to our previous work on scheduling of automated manufacturing systems. More specifically, multiresource job-shop with blocking was addressed in [18] with an approach similar to the one of this paper and in [19] with disjunctive approach. Multiresource shop scheduling with resource flexibility but without blocking was addressed in [21]. This paper addresses for the first time scheduling of multiresource shops with both resource flexibility and blocking. The main contributions of this paper are the following ones.

- *A general scheduling model* that integrates multiresource operations, resource flexibility and blocking.
- *Polynomial geometric approaches* to solve both the case of two flexible jobs and the case with one job without resource flexibility and one flexible job.
- *A heuristic for scheduling any number of jobs* that is based on the new polynomial geometric approaches, the notion of combined job proposed in [18] and a metaheuristic.
- *Extensive numerical results* that show the efficiency of the proposed heuristic.

The remainder of this paper is organized as follows. Section II formally defines the scheduling model and presents a mathematical formulation. Section III reviews relevant scheduling literature. In Section IV, we propose polynomial algorithms for the two-job cases. Section V describes a heuristic method for solving the scheduling model, which makes use of the polynomial algorithms. Numerical results are reported in Section VI. Section VII contains a conclusion and some suggestions for future research.

II. Problem Description and Modeling

This section presents the general scheduling model that covers many existing scheduling models with or without blocking constraints as special cases. The scheduling problem is defined as follows. A given set $\Im = \{J_1, J_2, \ldots, J_N\}$ of N jobs must be performed on a set $\Re = \{R_1, R_2, \ldots, R_m\}$ of m resources. Each resource is available in one copy and can be used for at most one operation at any time. Resources may include machines, transportation resources, machining tools, human operators, etc. The processing of a job J_i consists of several operations $\{O_{i1}, O_{i2}, \ldots, O_{i,n_i}\}$ to be performed in the given order, where n_i is the number of operations of job J_i. The operations are nonpreemptive, i.e., an ongoing operation cannot be interrupted. The problem is further characterized by the following features:

- *Multiresource operations*. Perform an operation $O_{i,j}$ needs simultaneously a subset of resources $A_{i,j} \subseteq \Re$. Thus, two operations $O_{r,s}$ and $O_{p,q}$ cannot be processed concurrently if they share common resources, i.e., $A_{r,s} \cap A_{p,q} \neq \emptyset$.
- *Multimode property*. The subset of resources $A_{i,j}$ to be used to perform an operation $O_{i,j}$ can be selected in a set $F_{i,j} = \{A^1_{i,j}, A^2_{i,j}, \ldots, A^{m_{i,j}}_{i,j}\}$ of $m_{i,j}$ alternative subsets also called candidate subsets. The processing time $p_{i,j}$ of operation $O_{i,j}$ is assumed to be independent of the alternative subset on which it will be performed. A job J_i is said a fixed job if the set $F_{i,j}$ of alternative subsets contains a singleton for all operation $O_{i,j}$ of J_i. Otherwise, the job is said a flexible job or simply a job.
- *Blocking constraint*. This feature concerns the release of resources after the completion of operations and requires that, after the completion of an operation $O_{i,j}$: i) all resources used for $O_{i,j}$ are held until all resources needed for the next operation $O_{i,j+1}$ of the same job become available and ii) then resources used for $O_{i,j}$ but not needed for $O_{i,j+1}$ are released and new resources needed for $O_{i,j+1}$ are held.

The scheduling problem consists of finding an assignment of alternative subsets to operations and of sequencing the operations on the resulting resources. The objective is to find a feasible solution (i.e., that satisfies precedence and resource capacity constraints, and without deadlock situations) in order to minimize the makespan.

It is known that a manufacturing system with complex product structures and complex resources behaviors may have deadlock states if the resources are not adequately managed. A system deadlock is a situation that arises when some in-process parts are involved in a circular waiting, i.e., each part waits for resources held by other parts in order to progress and release resources it holds. When this situation happens, related resource cannot be released, the circular wait situation will persist forever and the related parts can never complete. This problem has been neglected by the great majority of research on scheduling, which usually assume unlimited buffer capacity between resources. Deadlock prevention/avoidance is a critical scheduling and control problem of automated manufacturing systems [17]. For instance, in a robotic cell with a single robot a deadlock arises if the robot carries a part to a machine, which is working on another part. In this situation, the robot cannot load the part it carries since the machine is not free while the machine will need the robot to unload the part it is working on. Deadlock situations can also be found in railway scheduling if a train is on a track T_1 and goes to the next track T_2, while another train uses track T_2 and progresses to its next track T_1.

One remarkable feature of our scheduling model is its ability to represent a wide variety of practical settings. If we only consider the machine requirement of an operation $O_{i,j}$, then its resource requirement can be simply represented by $F_{i,j} = \{M(p_{i,j})\}$, where M is the related machine for $O_{i,j}$ and $p_{i,j}$ the processing time. If we want to further take into account human resource and tools, then $F_{i,j} = \{M \cap OP \cap T(p_{i,j})\}$, where OP is the competent operator for operation $O_{i,j}$ and T the machining tool needed for $O_{i,j}$. If the operator is only needed to mount the tool and to load the product but is not needed for the subsequent processing of the product by the machine, this situation can be represented by two consecutive operations $O_{i,j,1}$ and $O_{i,j,2}$ with $F_{i,j,1} = \{M \cap OP \cap T(\Delta_{i,j})\}$ and $F_{i,j,2} = \{M \cap T(p_{i,j})\}$, where $\Delta_{i,j}$ is the time needed to mount the tool and to load the product. Our model provides a generic framework for precise modeling of the following common product manufacturing phases: waiting in a buffer of unlimited capacity as $F_{i,j} = \{\emptyset(0)\}$, waiting in a buffer B of size n as $F_{i,j} = \{B_1 \cup B_2 \cup \cdots \cup B_n(0)\}$, where B_i denotes the ith storage unit of the buffer, transportation delay Δ as $F_{i,j} = \{\emptyset(\Delta)\}$, transportation with a robot R as

$F_{i,j} = \{R(\Delta)\}$. Further, by detailed modeling of wait between operations [18], classical models such as job-shops, flow-shops with or without blocking, production lines with buffers and robotic cells can be modeled as special cases of our model.

Even though resources are assumed to be different, resources available in multiple units can still be taken into account. This can be performed by distinguishing between identical resources and adequately defining the alternative ways of performing an operation. To see this, let $O_{i,j}$ be an operation that needs two resources R_1 and R_2, where the resource R_2 is available in three units. The situation can be modeled by subdividing R_2 in three resources $R_{2,1}$, $R_{2,2}$ and $R_{2,3}$. As a result, operation $O_{i,j}$ can be performed in three different ways on resources $\{R_1, R_{2,1}\}$, or on $\{R_1, R_{2,2}\}$ or on $\{R_1, R_{2,3}\}$.

The remainder of this section is devoted to the mathematical formulation of the scheduling model. This model will be used to evaluate the quality of the heuristic method proposed in this paper. As mentioned, the scheduling problem is to decide which alternative subset must be used to perform an operation and to find optimal starting times as well as the sequences on the resources. A scheduling solution is described by the following decision variables: $Y_{ij}^k = 1$ if operation $O_{i,j}$ is assigned to alternative subset k, $X_{ij,pq} = 1$ if operation $O_{i,j}$ is sequenced before $O_{p,q}$ and $S_{i,j}$ is the starting time of operation $O_{i,j}$. Using these variables, the makespan minimization can be formulated, as shown in (1)–(5) at the bottom of the page, where $M > 0$ is a large enough constant and $\epsilon > 0$ is a small enough constant.

In this formulation, constraints (1) ensure the precedence relations between operations of the same job, while constraints (2) guarantee that only one alternative subset is assigned to an operation. Constraints (5) are used to calculate the value of the makespan by using the starting time of the last operation of each job.

Particular attention is given, however, to constraints (3) and (4). These constraints contain both the blocking constraint and the disjunctive relations between two operations that use two alternative subsets with common resource. More precisely, if operations $O_{i,j}$ and $O_{p,q}$ are assigned to two alternative subsets that do not share any resource, then constraints (3) and (4) are redundant (since the quantity $2 - Y_{ij}^k - Y_{pq}^{k'}$ is positive) and the operations may be performed concurrently. However, if the alternative subsets A_{ij}^k and $A_{pq}^{k'}$ assigned to $O_{i,j}$ and $O_{p,q}$ share common resources (which is expressed by $A_{ij}^k \cap A_{pq}^{k'} \neq \emptyset$, $Y_{ij}^k = 1$ and $Y_{pq}^{k'} = 1$ in the mathematical model), then these operations must be performed one after another. Assume that operation $O_{i,j}$ is sequenced before $O_{p,q}$ (i.e., $X_{ij,pq} = 1$), then inequalities (3) and (4) become

$$S_{i,j} - S_{p,q+1} \geq \epsilon - M \tag{6}$$

$$S_{p,q} - S_{i,j+1} \geq \epsilon. \tag{7}$$

Therefore, inequality (6) is redundant and (7) requires that $O_{p,q}$ cannot be started before the starting of $O_{i,j+1}$, i.e., before operation $O_{i,j}$ progresses to its next operation and releases the resources needed to start $O_{p,q}$. This corresponds exactly to the definition of the blocking constraint.

The small positive number ϵ is necessary to ensure the deadlock-freeness of the schedules. To understand this, consider an example of two jobs and two resources: $J_1 = \{R_1(1)\}\{R_2(1)\}$ and $J_2 = \{R_2(1)\}\{R_1(1)\}$. The schedule $\{S_{1,1} = 0, S_{1,2} = 1, S_{2,1} = 0, S_{2,2} = 1\}$ fulfills the precedence constraints and the capacity constraints. In fact, constraints (1)–(4) hold with $\epsilon = 0$. However, the schedule leads to a deadlock at time 1. At this moment, job J_1 needs R_2 before releasing R_1, while job J_2 needs R_1 before releasing R_2. This is a deadlock situation. This example proves also that the precedence constraints and the resource capacity constraints do not imply the deadlock-freeness.

III. Literature Review

This section presents previous works on flow-shop and hybrid flow-shop with blocking, flexible job-shop, and scheduling problems taking into account deadlock situations. We do not present the literature on the multiresource operations. The reader is referred to [10] for a comprehensive survey. The flow-shop (resp. hybrid flow-shop) problem with blocking is addressed in [4] and [28] (resp. [29] and [31]). A comprehensive survey on scheduling problems with blocking and no-wait as well as applications of these constraints is given by [12]. Also, related to this paper is the rich body of the literature on part loading and scheduling of flexible manufacturing systems (FMS) (see, for instance, [14], [27], and [30]). A detailed review of FMS loading and scheduling is out of the scope of this paper

---

$$\begin{aligned}
\text{Minimize} \quad & C_{\max} \\
\text{subject to} \quad & S_{i,j+1} - S_{i,j} \geq p_{i,j} \; \forall i \in \Im, \quad \forall j = 1, \ldots, n_i - 1 && (1)\\
& \sum_{k=1}^{m_{ij}} Y_{ij}^k = 1 \quad \forall i \in \Im, \; \forall j = 1, \ldots, n_i && (2)\\
& S_{i,j} - S_{p,q+1} \geq \epsilon - M\left(2 - Y_{ij}^k - Y_{pq}^{k'} + X_{ij,pq}\right) \quad \forall O_{i,j}, O_{p,q}, k, k' | A_{ij}^k \cap A_{pq}^{k'} \neq \emptyset && (3)\\
& S_{p,q} - S_{i,j+1} \geq \epsilon - M\left(3 - Y_{ij}^k - Y_{pq}^{k'} - X_{ij,pq}\right) \quad \forall O_{i,j}, O_{p,q}, k, k' | A_{ij}^k \cap A_{pq}^{k'} \neq \emptyset && (4)\\
& C_{\max} \geq S_{i,n_i} + p_{i,n_i} \quad \forall i \in \Im \\
& X_{ij,pq}, Y_{ij}^k \in \{0,1\}, \quad S_{ij} \geq 0 && (5)
\end{aligned}$$

however let us note that there exists almost none scheduling models taking into account both blocking and flexibility, two important features of FMS.

Research on the flexible job-shop began with the work of [3], which developed a polynomial-time algorithm for solving the problem with two jobs in which the processing time of an operation does not depend on the selected machine. The NP-hardness of the two-job scheduling problem was proved in [20] for the general case where the processing time depends on the machine to be selected, and polynomial-time algorithms were proposed for the case with only one flexible job. Problems with realistic sizes are tackled using heuristics of two types: hierarchical approaches and integrated approaches. In the former type [25], the machine assignment and machine sequencing subproblems are separated. The later type [8] solves simultaneously the assignment and sequencing subproblems. These heuristics are based on taboo search and disjunctive graph model and address the makespan minimization.

The literature on scheduling problems with both multiresource operations and flexibility on the realization of operations is scarce. Taboo search techniques using the disjunctive graph model were proposed in [2], [9]. Recently, another heuristic based on a greedy heuristic and a genetic algorithm was proposed in [21]. Their model considers also the case where resources may be available in multiple units. Multimode and multiresource operations have also been investigated in multimode resource-constrained project scheduling literature.

Consider now the literature on deadlock-free scheduling. Some research efforts have been directed toward developing exact and heuristic methods based on Petri nets. The basic idea of the solution approaches is to first represent the behavior of the systems by using Petri net models, and then searching the reachability graph for the optimal firing sequence, using the A^* algorithm. An approach of this type is used in [32] for scheduling semiconductor plants. The scheduling problem of an automated manufacturing workstation was considered in [26] under the assumption of limited buffer capacity and the use of a robot is for material handling. The job-shop problem with blocking and no-wait constraints is addressed in [16]. The authors used several techniques from the literature of classical job-shop scheduling to develop heuristics and exact algorithms for obtaining deadlock-free schedules. The job-shop problem with blocking has been also tackled in [22] using a rollout metaheuristic.

The multiresource job-shop problem with blocking is investigated in [6] and [7]. In the former paper, the authors assumed that each resource is available in one unit and developed a heuristic approach based on a dynamic programming approach and Petri nets. The second paper extends the results developed in the former paper, for the case where a resource may be available in multiple units. The multiresource job-shop problem with blocking was further analyzed in [18] for the case where a resource is available in multiple units. A heuristic algorithm that schedules jobs sequentially according to a job sequence was proposed. Our work [19] is the first that attempts to find deadlock-free schedules using local search methods. The multiresource job-shop with blocking in which each resource is available in a single unit was modeled using an extension of the classical disjunctive graph, and a taboo search heuristic was used to find near-optimal solutions.

IV. Geometric Approach for Scheduling Two Flexible Jobs

When there is no flexibility and both jobs are fixed jobs, the geometric approach proposed in [18] directly applies and Section IV-A presents a variant of the algorithm better suited for extension to flexible jobs. When at least one job is flexible, the geometric approach does not directly apply and exhaustive enumeration of all possible resource assignment solves the problem but leads to an algorithm of exponential complexity. To overcome this difficulty, in Section IV-B we exploit the properties of geometric approach and prove that the two-flexible job problem can be transformed into a shortest path problem in a network. Section IV-C further improves this approach for the case of one flexible job and one fixed job.

A. Geometric Approach for Scheduling Two Fixed Jobs

The difference between the algorithm below and the one of [18] lies in deadlock situations that were explicitly represented in the geometric representation of the problem in [18], while they are neither determined nor considered here. The geometric approach consists of: i) representing the scheduling problem as a shortest path problem in a $2D$-plane and ii) transforming the problem into a shortest path problem. We use an illustrative example of two jobs J_1 and J_2 and four resources R_i with $i = 1, \ldots, 4$. Job J_2 has three operations and requires first R_4 for two time units, then R_1 for one time unit, and finally both R_2 and R_4 for four time units. Notation $J_2 = \{R_4(2)\}\{R_1(1)\}\{R_2R_4(4)\}$ is used. The first job is defined as $J_1 = \{R_2(4)\}\{R_1(2)\}\{R_3(2)\}$.

1) Representation of the Scheduling Problem on a $2D$-Plane: It consists of: representing each job J_i on an axis with n_i intervals each corresponding to an operation $O_{i,j}$ of length of $p_{i,j}$, and representing resource conflict between operations of the two jobs by obstacles. A rectangle, denoted (k_1, k_2), of the $2D$-plane corresponding to O_{1,k_1} and O_{2,k_2} forms an obstacle if they share common resources. Any schedule of the two jobs corresponds to a path from the origin O and the final point F with three types of segments: horizontal (J_1 only), vertical (J_2 only) and diagonal (both J_1 and J_2). The length of a horizontal/vertical segment is its natural length. However, the length of a diagonal segment is equal to its projection on either axis which is the time needed for simultaneous processing of both J_1 and J_2. A path from O to F is feasible if it avoids: i) the interior of an obstacle $(k_1,\ k_2)$ representing resource conflict; ii) the upper boundary of an obstacle $(k_1,\ k_2)$ with $k_2 < n_2$ as it corresponds to a situation in which O_{1,k_1} of J_1 is in progress and J_2 has finished O_{2,k_2} but does not yet start its next operation which, due to the blocking constraint, is also a resource conflict state; iii) the right boundary of an obstacle $(k_1,\ k_2)$ with $k_1 < n_1$; and iv) the North-East (NE) corner of any deadlock state $(k_1,\ k_2)$ for which both $(k_1,\ k_2 + 1)$ and $(k_1 + 1,\ k_2)$ are obstacles. Fig. 1 shows a feasible path and its graphical description.

2) Construction of the Network: The main step of the geometric approach is the construction of a network $N_0 = (V_0, E_0, d_0)$, where V_0 denotes the set of nodes, E_0 the set of directed arcs and d_0 the distance function of the arcs. It

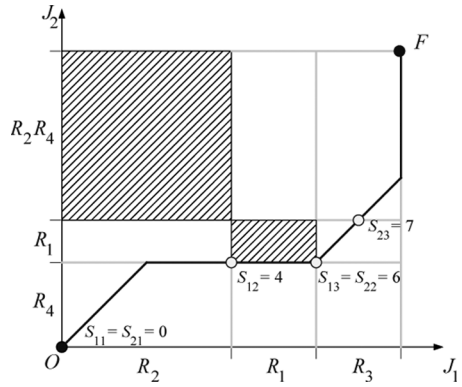

Fig. 1. A feasible path and its graphical description.

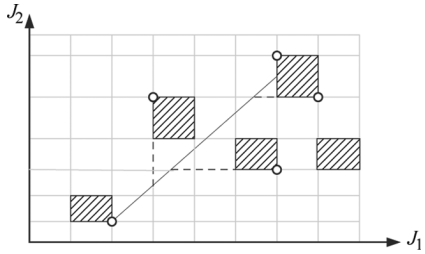

Fig. 2. Successors of a node.

transforms the shortest path problem in the $2D$-plane into the one of a shortest path in the network. The nodes of the network include the origin O, the final point F and the North-West (NW) and South-East (SE) corners of all obstacles $(k_1,\ k_2)$, denoted as $NW(k_1,\ k_2)$ and $SE(k_1,\ k_2)$.

The set of arcs or successors are defined as follows. Let us start with a node $SE(k_1,\ k_2)$ corresponding to a state in which O_{1,k_1} and O_{2,k_2-1} are finished. J_2 cannot progress first and J_1 must progress first as the right boundary of the rectangle $(k_1,\ k_2)$ is forbidden. This is not possible and $SE(k_1,\ k_2)$ is a deadlock state if $(k_1+1,\ k_2-1)$ is an obstacle (since the progress on the upper boundary is forbidden). This implies that a node $SE(k_1,\ k_2)$ with $(k_1+1,\ k_2-1)$ as an obstacle does not have successors. Similarly, nodes $NW(k_1,\ k_2)$ with $(k_1-1,\ k_2+1)$ as an obstacle does not have successors. For any other node v_i, all nodes that can be reached by first going diagonally and then vertically (resp. horizontally), are considered as successors of v_i (see Fig. 2). Immediate vertical (resp. horizontal) progress is forbidden at $SE(k_1,\ k_2)$ with $k_1 < n_1$ (resp. $NW(k_1,\ k_2)$ with $k_2 < n_2$). Note that it is not necessary to include in V_0 the following nodes: $SE(k_1,\ k_2)$ with $(k_1{+}1,\ k_2{-}1)$ as an obstacle, $NW(k_1,\ k_2)$ with $(k_1{-}1,\ k_2{+}1)$ as an obstacle.

The length $d_0(v,v')$ of any arc (v,v') with $v=(x,y)$ and $v'=(x',y')$ is equal to the projection along axis X (resp. Y) if v' is obtained from v by first going diagonally and then horizontally (resp. vertically).

The network of the illustrative example is given in Fig. 3. The path $(O, SE(2,2), F)$ is a shortest path of length 11 corresponding to perform J_1 and J_2 in parallel until instant $t=2$, perform J_1 while holding the resource of $O_{2,1}$, perform J_1 and J_2 in parallel between $t=6$ and $t=8$, and then finish with the last operation $O_{2,3}$ of J_2.

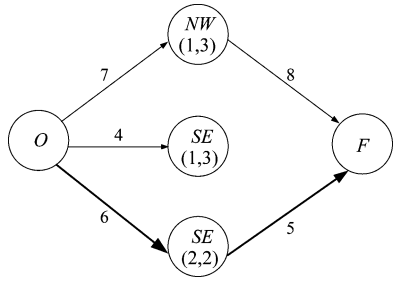

Fig. 3. Network of the illustrative example.

Theorem 1: A shortest path from O to F in the constructed network N_0 corresponds to a shortest path on the $2D$-plane with obstacles. Hence, it corresponds to an optimal schedule.

Proof: Notice first that any path from O to F in N_0 is a feasible path as nodes corresponding to deadlock situations do not have any successor. The proof is based on the relation between the network N_0 constructed here and the network N^* used in [18] that differs by explicitly including deadlock obstacles. It is proved in [18] that there exists an optimal path P^* corresponding to a series of m subpaths $SP^*(s_i, s_{i+1})$, where $s_0=O$, $s_m=F$ and other nodes s_i are NW/SE corners of resource/deadlock-obstacles. We will prove that this path can be obtained in N_0. Since the progress from any node, that consists of going first diagonally and then horizontally/vertically, is the same in the two networks and the path P^* must avoid the interior, the upper boundary and the right boundary of all resource-obstacles then the path P^* is feasible in N_0. It is enough to prove that the path P^* can be represented by a series of l subpaths $SP(c_i, c_{i+1})$, where $c_0=O$, $c_l=F$ and other nodes c_i are NW/SE corners of resource-obstacles. Consider the case where s_j is the NW/SE corner of a deadlock-obstacle $(k_1,\ k_2)$. Without loss of generality, let s_j be the SE corner of $(k_1,\ k_2)$. From the definition of a deadlock-obstacle $(k_1,\ k_2)$ in [18], rectangle $(k_1+1,\ k_2)$ is a deadlock/resource-obstacle. The right boundary of $(k_1,\ k_2)$ needed for vertical progress and the interior of $(k_1+1,\ k_2)$ needed for diagonal progress are both forbidden in the construction of P^*. As a result, P^* can only progress horizontally at point s_j and this continues as long as P^* does not hit the SE corner of a resource-obstacle. ■

Proposition 1 [18]: The complexity of the shortest path algorithm is $O(n_1\ n_2\ (n_1+n_2))$.

B. Geometric Approach for Scheduling Two Flexible Jobs

There is no straightforward $2D$-plane representation of the scheduling problem of two flexible jobs. When the resource assignment is given, i.e., an alternative subset $A^r_{i,j}$ is chosen for each operation $O_{i,j}$, a $2D$-representation can be given, a network can be constructed and the optimal schedule can be determined from the shortest path of the network. The approach we propose for scheduling two flexible jobs consists of constructing a network that contains as subgraph any network related to any resource assignment. As a result, the optimal resource assignment and its optimal schedule can be determined from the network. The remaining part of this subsection is organized as follows: (A) a $2D$-plane representation; (B) construction of the network; (C) improvement of the network; and (D) complexity of the two-flexible-job problem.

1) A 2D-Plane Representation of the Scheduling Problem: The $2D$-representation is the same as for the representation of two-fixed-job problem except that it is not possible to define obstacles. Instead, we use the concept of potential obstacles defined as follows.

Definition 1: A rectangle $(k_1,\ k_2)$ is said to be a potential obstacle if there exist alternative sets A^r_{1,k_1} and A^s_{2,k_2} that share common resources, i.e., $\exists A^r_{1,k_1} \in F_{1,k_1}$ and $\exists A^s_{2,k_2} \in F_{2,k_2}/A^r_{1,k_1} \cap A^s_{2,k_2} \neq \emptyset$. It is considered as a real obstacle if $A^r_{1,k_1} \cap A^s_{2,k_2} \neq \emptyset, \forall A^r_{1,k_1}, A^s_{2,k_2}$.

Contrary to the $2D$-representation of the two-fixed-job problem that contains all information needed for scheduling, the new $2D$-representation does not contain information about resource assignment. Thus, a path crossing a potential obstacle on the $2D$-plane is forbidden only if the related operations are assigned to common resources. Otherwise, such a path is not forbidden.

2) Constructing a Network: The basic idea is to construct a network $N_1 = (V_1,\ E_1,\ d_1)$ that encapsulates as subgraphs the networks N_0 of all possible resource assignments. Let us first present an approach that will later be improved. The network N_1 is defined as follows.

The set V_1 of nodes includes O, F, $SE(k_1,\ k_2,\ A^r_{1,k_1},\ A^s_{2,k_2-1})$, and $NW(k_1,\ k_2,\ A^r_{1,k_1-1},\ A^s_{2,k_2})$ for all potential obstacles $(k_1,\ k_2)$ and for all alternative sets $A^r_{1,k_1}, A^s_{2,k_2-1}, A^r_{1,k_1-1}$ and A^s_{2,k_2}. Subsets A^r_{1,k_1} and A^s_{2,k_2-1} (resp. A^r_{1,k_1-1} and A^s_{2,k_2}) give alternative subsets used to reach the SE-corner (resp. NW-corner) and keep all information needed to determine future evolution.

The set E_1 of arcs is defined as follows. There exists an arc from node v to v' if it is possible to connect v to v' on the $2D$-plane by first progressing diagonally then horizontally/vertically with appropriate resource assignment for all operations encountered during the progress. The progress must start at v with alternative sets associated with v and end at v' with alternative set associated with v'. As the right and upper boundaries of rectangle $(k_1,\ k_2)$ are forbidden if $(k_1,\ k_2)$ is a real obstacle, immediate vertical (resp. horizontal) progress at v is forbidden if $v = SE(k_1,\ k_2,\ A^r_{1,k_1},\ A^s_{2,k_2-1})$ with $k_1 < n_1$ (resp. $v = NW(k_1,\ k_2,\ A^r_{1,k_1-1},\ A^s_{2,k_2})$ with $k_2 < n_2$). As a result, the path from v to v' starts with the progress of J_1 (resp. J_2) if $v = SE(k_1,\ k_2,\ A^r_{1,k_1},\ A^s_{2,k_2-1})$ (resp. $v = NW(k_1,\ k_2,\ A^r_{1,k_1-1},\ A^s_{2,k_2})$). Details of the progress on the $2D$-plane will be explained in the Appendix.

The length $d_1(v,\ v')$ of an arc $(v,\ v')$ is the same as for two-fixed-job case, since the processing time of operations does not depend on the assignment.

Theorem 2: Any path from O to F of the network N_0 for any resource assignment is a path of the network N_1. Further, any path from O to F of the network N_1 corresponds to at least a feasible path from O to F in a $2D$-plane related to some resource assignment.

Proof: Two types of $2D$-plane representations are considered here: the $2D$-plane representation for two fixed jobs that we denote as $2D$-fixed and the $2D$-plane representation of two flexible jobs that we denote $2D$-flexible.

Consider first a resource assignment ξ with $A^{r_{i,j}}_{i,j}$ as the alternative set selected for any $O_{i,j}$. Consider a path of N_0 such that $P = SP(s_0, s_1), SP(s_1, s_2), \ldots, SP(s_{m-1}, s_m)$ with $s_0 = O$, $s_m = F$ and s_i with $0 < i < m$ being SE/NW corners of real obstacles of the $2D$-fixed representation related to resource assignment ξ. By construction, if s_i is a SE (resp. NW) corner of an obstacle (k_1, k_2), then it is a node $SE(k_1, k_2, A^{r_{1,k_1}}_{1,k_1}, A^{r_{2,k_2}-1}_{2,k_2-1})$ (resp. $NW(k_1, k_2, A^{r_{1,k_1}-1}_{1,k_1-1}, A^{r_{2,k_2}}_{2,k_2})$) in N_1. Further, it is possible to reach node s_i and progress in the $2D$-flexible representation from that node to s_{i+1} by simply following resource assignment ξ. As a result, P is also a path in N_1. Conversely, consider a path of N_1 such that $P = SP(c_0, c_1), SP(c_1, c_2), \ldots, SP(c_{m-1}, c_m)$ with $c_0 = O$, $c_m = F$ and c_i with $0 < i < m$ being nodes of form $SE(k_1, k_2, A^r_{1,k_1}, A^s_{2,k_2-1})$ or $NW(k_1, k_2, A^r_{1,k_1-1}, A^s_{2,k_2})$ of the $2D$-flexible representation. Without loss of generality, let $c_i = SE(k_1, k_2, A^r_{1,k_1}, A^s_{2,k_2-1})$ and $c_{i+1} = SE(k'_1, k'_2, A^{r'}_{1,k'_1}, A^{s'}_{2,k'_2-1})$. By construction, $SP(c_i, c_{i+1})$ is a path connecting c_i to c_{i+1} by starting with alternative subsets $(A^r_{1,k_1}, A^r_{2,k_2-1})$ at c_i, ending with alternative subsets $(A^{r'}_{1,k'_1}, A^{s'}_{2,k'_2-1})$ and using appropriate resource assignment for intermediate operations. Repeating the argument for all subpaths $SP(c_i, c_{i+1})$ defines a complete resources assignment ξ such that P is feasible in the related $2D$-fixed representation. This completes the proof. Note that, the nodes c_i need not to be a node in the related network N_0 as the corresponding potential obstacles need not to be real obstacles. ■

Combining Theorems 1 and 2, we obtain the following result.

Theorem 3: The shortest path from O to F of the network N_1 corresponds to an optimal schedule of the two-flexible-job problem.

Remark 1: Theorem 3 allows the determination of the starting times of operations. The corresponding resource assignment will be addressed in the Appendix.

3) Improvement of the Network: Consider the SE/NW corners of the network N_1. As mentioned above, starting from a node $v = SE(k_1, k_2, A^r_{1,k_1}, A^s_{2,k_2-1})$, immediate vertical progress is forbidden and J_1 must progress before J_2 at v. This means that operation O_{1,k_1+1} will start first, while J_2 still holds resource subset A^s_{2,k_2-1}. Any alternative set $A^{r'}_{1,k_1+1}$ such that $A^{r'}_{1,k_1+1} \cap A^s_{2,k_2-1} \neq \emptyset$ can be chosen for O_{1,k_1+1}. As a result, the set of all feasible alternative subsets for O_{1,k_1+1} depends on A^s_{2,k_2-1} but is independent of A^r_{1,k_1}. After the start of operation O_{1,k_1+1}, operation O_{2,k_2} might start as well and the alternative subset $A^{s'}_{2,k_2}$ to be selected depends on the alternative subset selected for O_{1,k_1+1}. To summarize, possible evolution and selection beyond the corner $SE(k_1, k_2, A^r_{1,k_1}, A^s_{2,k_2-1})$ is totally independent of A^r_{1,k_1}. We then neglect this information in the definition of SE corners and represent them as $SE(k_1, k_2, *, A^s_{2,k_2-1})$. Similarly, NW corners can be represented as $NW(k_1, k_2, A^r_{1,k_1-1}, *)$.

The above changes result in a more compact network $N_2 = (V_2, E_2, d_2)$ defined similarly as N_1 but with V_2 containing O, F, $SE(k_1, k_2, *, A^s_{2,k_2-1})$ and $NW(k_1, k_2, A^r_{1,k_1-1}, *)$ of all potential obstacles. It can be proved that Theorems 2 and 3 hold with N_1 replaced by N_2.

4) The Scheduling Algorithm and its Complexity: Based on above presentations, the scheduling problem of two flexible jobs can be summarized in Algorithm A1. The algorithm considers only the SE and NW corners of potential obstacles that can

be reached in the plane. In Step 2, if the set $ADJACENT$ is empty, this means that the immediate progress starting from v^0 is not possible and node v^0 has no outgoing arc.

Algorithm A1. Scheduling Two Flexible Jobs:

Step 1: Initialize $REACHED \leftarrow \{O\}$.

Step 2: While $REACHED$ is not empty:

— select a node $v^0 \in REACHED$ and apply the Algorithm A3 in the Appendix to obtain the set $ADJACENT$ of nodes obtained from v^0 by first going diagonally then horizontally/vertically with appropriate resource assignment;

—$REACHED \leftarrow REACHED \cup ADJACENT$.

Step 3: Determine the length of each arc in E_2. This completes the construction of N_2.

Step 4: Determine the shortest path P^* from O to F of N_2.

Step 5: Apply Algorithm A6 in the Appendix to determine for each subpath of P^* its complete resource assignment.

Theorem 4: The scheduling problem of two flexible jobs can be solved in $O((T_1 + T_2)(T_1 n_1 + T_2 n_2)\ n_1 n_2)$ time, where $T_i = \max_{k=1..n_i} |F_{i,k}|$.

Proof: The complexity of the Algorithm A1 is mainly due to Steps 2 and 4. The number of successors of a node $v_i = (k_1, k_2, *, *)$ depends on the processing times of operations. An upper bound on the number of successors of v_i is $(T_1 n_1 + T_2 n_2)$, where $T_i = \max_{k=1..n_i} |F_{i,k}|$. Indeed, the diagonal progress can cross $(n_1 - k_1)$ (resp. $(n_2 - k_2)$) right (resp. upper) boundaries before reaching the final point F. Each vertical V_k (resp. horizontal H_k) progress can generate $|F_{1,k}|$ (resp. $|F_{2,k}|$) nodes. This implies that the total number of nodes that are considered as successors of v_i is bounded by $(T_1 n_1 + T_2 n_2)$. Further, there are at most $(T_2\ n_1\ n_2)$ SE nodes and $(T_1\ n_1\ n_2)$ NW nodes which correspond to all potential obstacles. As a result, Step 2 of the construction of the network takes $O((T_1 + T_2)(T_1 n_1 + T_2 n_2)\ n_1 n_2)$ time. Since the network is acyclic, Step 4 of the determination of the shortest path takes $O(|V_2| + |N_2|)$. Consequently, the overall complexity of the Algorithm A1 is $O((T_1 + T_2)(T_1 n_1 + T_2 n_2)\ n_1 n_2)$. ∎

C. Geometric Approach for Scheduling One Flexible Job and One Fixed Job

Let J_1 be a flexible job and J_2 be a fixed job. Of course, this is a special case of two flexible jobs. However, due to the importance of this case in scheduling N jobs, we propose a dedicated algorithm that is more efficient.

Consider first the network N_2 of this case. According to its definition, N_2 contains the following nodes: O, F, $SE(k_1, k_2, *)$ and $NW(k_1, k_2, A^r_{1,k_1-1})$ as J_2 is a fixed job. Our improvement concerns nodes $NW(k_1, k_2, A^r_{1,k_1-1})$. Note that, at $NW(k_1, k_2, A^r_{1,k_1-1})$, job J_2 must progress before job J_1. The information A^r_{1,k_1-1} is necessary only to determine whether vertical progress can pursue. However, if the path chooses diagonal progress at $NW(k_1, k_2, A^r_{1,k_1-1})$, then any candidate in $FEAS(F_{1,k_1}, A_{2,k_2+1})$ can be chosen for the operation O_{1,k_1} of J_1, where $FEAS(F, A) = \{A' \in F \mid A' \cap A = \emptyset\}$ is used to indicate the set of alternative subsets belonging to F that do not share common resources with alternative subset A. This implies that diagonal progress at $NW(k_1, k_2, A^r_{1,k_1-1})$ is independent of A^r_{1,k_1-1}. What is important for the diagonal progress is to check the possibility of reaching a NW corner of type $NW(k_1, k_2, A^r_{1,k_1-1})$ and then starting O_{2,k_2+1}.

To take advantage of this, we define a new network $N_3 = (V_3, E_3, d_3)$ constructed similarly as N_2 except the following.

1) V_3 contains O, F, $NW(k_1, k_2, *)$ such that $FEAS(F_{1,k_1}, A_{2,k_2+1}) \neq \emptyset$, $SE(k_1, k_2)$ for all potential obstacles (k_1, k_2).
2) When progressing from node $v_0 = NW(k_1, k_2, *)$ using Algorithm A3 of the Appendix, immediate vertical/horizontal progress is forbidden and only diagonal progress is allowed initially.
3) When moving vertically after a diagonal progress in Step 5 of Algorithm A3, the vertical move must not stop at the first NW corner encountered but must continue till all NW corners reachable along the same vertical have been identified.
4) A node $v = NW(k_1, k_2, *)$ met during a vertical progress is considered as successor of a node v_0 only if it is possible to go from v_0 to $NW(k_1, k_2, *)$ and then progress diagonally, i.e., if v is a node of N_3 and if diagonal progress is possible after reaching v, i.e., $FEAS(H, A_{2,k_2+1}) \neq \emptyset$.

Remark 2: Step 3 above guarantees that vertical moves between NW corners of the same vertical have been taken into account and the only progress starting from NW corners is the diagonal progress. The information A^r_{1,k_1-1} is not needed for defining a node related to a NW corner.

Remark 3: The number of nodes of the network N_3 is at most equal to $n_1\ n_2$, and the number of arcs is significantly decreased. Therefore, the complexity of Algorithm A1 is improved.

V. The Heuristic Method for Scheduling N Jobs

Our approach for scheduling N jobs combines metaheuristics and the results of the previous section on two-job cases. It relies on: i) a greedy algorithm that schedules jobs according to a given permutation π of the set of jobs $\{J_1, J_2, \ldots, J_N\}$ and ii) a metaheuristic to identify the best permutation. The permutation is used by the greedy algorithm for building the complete schedule and should not be confused with the input sequence of jobs into each resource. The metaheuristic is used to identify the permutation with which the greedy algorithm performs the best.

A. Greedy Algorithm

For each given permutation π, the greedy methodology of [18] is adapted to generate a complete schedule. It consists of scheduling the jobs one after another according to the permutation π. At the beginning the first two jobs are scheduled optimally using the algorithm of Section IV for two flexible jobs. The resource assignment and the sequences on resources of these two jobs are then fixed. The schedule is then represented by a *combined job* introduced in [18], which is defined according to the resource requirement along the time. This combined job is then considered as a simple fixed job. Scheduling algorithm for one fixed job and one flexible job is used to schedule the combined job and the third job in permutation π. A new combined job is then defined to represent the schedule of the three jobs. This process continues till the schedule of all jobs. More precisely, the greedy heuristic is summarized in Algorithm A2.

Algorithm A2. Greedy Heuristic for Scheduling N Jobs:

Step 1: Let $J_{[1]}, J_{[2]}, \ldots, J_{[N]}$ be a permutation of jobs.
Step 2: Schedule $J_{[1]}$ and $J_{[2]}$ optimally using Algorithm A1.
Step 3: Construct J_{com} of the resulting schedule.
Step 4: For $i = 3$ to N:
— schedule J_{com} and $J_{[i]}$ using the improved version of Algorithm A1 with J_{com} as a simple fixed job;
— construct the new J_{com} of the resulting schedule.
Step 5: Derive the final schedule from J_{com}.

The idea behind the combined job is very simple and is explained as follows. Let C_m be the makespan of the optimal schedule of a subset of jobs. The time interval $[0, C_m]$ is then decomposed into subintervals according to the starting/finishing times of operations, and each subinterval is associated to an operation of the combined job. Hence, the number of operations of the combined job is equal to the number of subintervals. The processing time of an operation $O_{com,j}$ is equal to the length of the related subinterval and the resources needed by this operation are the resources hold by all jobs in the corresponding subinterval.

The major difference with [18] is the consideration of the combined job as a simple job and hence the use of algorithms of the previous section to schedule a combined job. This contradicts with the need in [18] to treat a combined job as a special job and the need of special algorithms to schedule a combined job and a simple job. The correctness of the new greedy algorithm is mainly due to the assumption of non identical resources and will be proved in Theorem 5.

Remark 4: Once the combined job is constructed, the sequence of operations on the resources of the related jobs will not be changed in the next iterations. As a result, a combined job is considered as a simple fixed job without resource flexibility. The blocking constraint of the combined job ensures that an operation $O_{i,j}$ of a job split in two consecutive operations $O_{com,k}$ and $O_{com,k+1}$ of the combined job does not release its resources at the end of $O_{com,k}$. Therefore, the nonpreemption of operations is satisfied.

Remark 5: The permutation π of Algorithm A2 is the order in which jobs are taken into account in the greedy algorithm. It should not be confused with the order in which jobs are performed. It is possible that a job $J_{[i+1]}$ is started and completed before job $J_{[i]}$ preceding $J_{[i+1]}$ in the permutation π.

Deriving a schedule from a combined job is straightforward by simply choosing for each operation $O_{i,j}$ of a job the starting time of the corresponding combined operation in which operation $O_{i,j}$ first appears.

Theorem 5: The schedule of Algorithm A2 is feasible.

Proof: By definition of the combined job, it is obvious that the corresponding schedule meets the resource capacity constraints and the precedence constraints. The blocking constraint of the combined job further ensures the nonpreemption constraints of the operations. The only remaining issue to check is whether all operations $O_{i,j}$ scheduled to start at any given time t can start without violating the blocking constraints of the related jobs.

The remaining part concerns the proof of the feasibility with respect to blocking constraints of jobs. Let $J_{com[i]}$ be the combined job obtained at iteration i. We prove by induction on $i = 2, 3, \ldots, N$ that:

(P) *all new operations $O_{j,k}$ of operation $O_{com[i],k'}$ can start at the end of $O_{com[i],k'-1}$.*

Property (P) clearly holds for $i = 2$ as the schedule given by Algorithm A1 of two flexible jobs is feasible. Assume that (P) is true for i. Consider the schedule between $J_{[i+1]}$ and $J_{com[i]}$. To prove (P) for $i+1$, it is enough to show that, for any time t when one of the two jobs $J_{[i+1]}$ and $J_{com[i]}$ move to its next operation, all new operations $O_{j,k}$ can indeed start immediately after t. Four cases are possible: i) only $J_{[i+1]}$ moves from $O_{[i+1],k}$ to $O_{[i+1],k+1}$; ii) only $J_{com[i]}$ moves from $O_{com[i],k'}$ to $O_{com[i],k'+1}$; iii) $J_{[i+1]}$ moves first from $O_{[i+1],k}$ to $O_{[i+1],k+1}$ and $J_{com[i]}$ moves second from $O_{com[i],k'}$ to $O_{com[i],k'+1}$; and iv) $J_{com[i]}$ moves first from $O_{com[i],k'}$ to $O_{com[i],k'+1}$ and $J_{[i+1]}$ moves second from $O_{[i+1],k}$ to $O_{[i+1],k+1}$. For case i), resources needed by $O_{[i+1],k+1}$ are available at time t and move i) is feasible. For case ii), resources needed for new operations $O_{j,k}$ related to $O_{com[i],k'+1}$ are not hold by job $J_{[i+1]}$. By induction assumption, move ii) is feasible. For case iii), according to the feasibility of the schedule of $J_{[i+1]}$ and $J_{com[i]}$, resources needed for $O_{[i+1],k+1}$ are not hold by $O_{com[i],k'}$ and hence not hold by operations $O_{j,k}$ related to $O_{com[i],k'}$. As a result, the move from $O_{[i+1],k}$ to $O_{[i+1],k+1}$ is feasible. After the move from $O_{[i+1],k}$ to $O_{[i+1],k+1}$, according to the feasibility of the schedule of $J_{[i+1]}$ and $J_{com[i]}$, resources needed for new operations $O_{j,k}$ related to $O_{com[i],k'+1}$ are not hold by $O_{[i+1],k+1}$ and hence available. By induction assumption, the move from $O_{com[i],k'}$ to $O_{com[i],k'+1}$ is feasible. Thus, the two moves of case iii) are both feasible. Similarly, the two moves of case iv) are feasible. To summarize, property (P) holds for $i+1$. Hence, by induction, property (P) holds. ■

B. Improving the Greedy Algorithm

This subsection aims at identifying the permutation of jobs with which the greedy algorithm performs the best. Many metaheuristics can be used to optimize the permutation of jobs and three techniques are compared in this paper: a random method (RND), a taboo search (TS) and a genetic algorithm (GA). The implementation of these techniques is fairly standard and is described as follows.

The random method starts by putting all jobs in a set, which represents jobs not yet selected. At each step a job is randomly selected from the set and moved to a list that is initially empty. The selected job is inserted at the end of this list. The random selection continues until the set becomes empty, and the sequence of jobs in the list represents the permutation of jobs from which the greedy heuristic is applied.

The taboo search is started from an initial permutation π_0 of jobs obtained as follows. The polynomial algorithm of Section IV is applied to schedule any possible couple of jobs. The two jobs leading to a schedule of shortest makespan with ties broken randomly are selected as the first two jobs of π_0. A combined job representing the schedule of these two jobs is then constructed. The scheduling problem of the combined job and each of the remaining jobs is then solved. Accordingly, the job that gives the shortest makespan is the third job. This process continues until all the jobs are examined, and the order

TABLE I
NUMBER OF TIMES A METHOD GIVES THE BEST MEAN MAKESPAN

$model$	$size$	after 5 secs			after 60 secs			after 600 secs		
		RND	TS	GA	RND	TS	GA	RND	TS	GA
$JSPB$	10×5	3	1	1	1	2	2	2	5	5
	10×10	0	3	2	1	2	3	3	3	2
	15×10	1	4	0	0	4	1	0	2	3
	20×10	1	2	2	1	2	2	1	2	2
	Total	5	10	5	3	10	8	6	12	12
$JMPMB$	10×5	2	2	1	1	2	2	3	5	5
	10×10	0	2	3	0	2	3	1	3	2
	15×10	1	3	1	1	2	2	0	3	2
	20×10	1	2	3	0	1	4	1	2	2
	Total	4	9	8	2	7	11	5	13	11

on which the jobs are selected gives the initial solution π_0 of the taboo search. Given a permutation $\pi = J_{[1]}\, J_{[2]} \ldots J_{[N]}$, a neighbor solution is obtained a move that selects a job at position x, i.e., job $J_{[x]}$, and inserts it at some position $y \neq x$. The number of neighbors is equal to $(N-1)^2$, which increases with the size of the problem. Each neighbor solution p is evaluated by the greedy algorithm with p as the job sequence. In an attempt to make a large perturbation on the actual solution and reduce the computational time, we restrict the distance of a move defined as $|x-y|$ to be at least $\lceil N/3 \rceil$. The attributes of a move of a job J_i from position x to position y recorded in the taboo list are (J_i, x). As long as the couple (J_i, x) remains in the taboo list, job J_i is forbidden from returning to position x. A dynamic taboo list is implemented and, every $2N$ iterations, a new size of the taboo list is randomly selected in the interval $[0.9N, 1.3N]$. In order to override the taboo status of a move, the global aspiration that selects a taboo move if the corresponding neighbor solution has a smaller makespan than the makespan of the best solution obtained so far, is used.

The genetic algorithm starts from an initial population constructed in a very simple way by randomly generating the chromosomes (like in the random method). The size of the population is set experimentally to 20 chromosomes. The indirect encoding in which a chromosome contains an n-string corresponding to a permutation of all integers from 1 to n, is used. The population replacement is used in our application since it allows a high population diversity, allowing the greedy heuristic to start from diverse chromosomes. For selecting pairs of parents to be crossed, the simplest and most used roulette wheel selection is applied, which uses a distribution on the current population that depends on the fitness of the solutions. The partially mapped crossover [23] that has been applied successfully on many scheduling problems and more specifically on shop problems, is selected. After using the crossover operator, a mutation operator is applied that deletes a randomly selected job from its current position in the chromosome and inserts it at another randomly selected position. The crossover operator is applied with a probability equal to 0.9, and the mutation probability is equal to 0.05.

The three algorithms are compared on 40 instances with different sizes including 20 instances of the job shop problem with blocking $(JSPB)$ and 20 instances of multipurpose job shop problem with blocking $(JMPMB)$. For each instance, five independent runs of each algorithm are performed. The measure of performance adopted in the comparison is the mean makespan obtained after the five independent runs. Table I gives the number of times a method gives the best mean makespan for the selected instances. The mean makespan obtained after 5, 60, and 600 s are given in Columns 3, 4, and 5, respectively. From these columns, we observe that the random method obtains satisfactory results but they are of lower quality compared with the taboo search and the genetic algorithm. After 5 s results given by the random method are comparable with the two other methods but the quality of the random method diminishes as the execution time increases. Further, no method outperforms the other methods on all the instances. However, the taboo search and the genetic algorithm appear equivalent in particular when the execution time increases. Note that when a method outperforms the other the relative error of the makespan obtained by a method over the other method is small. Recall again that these results and conclusions can change if other parameters are chosen for the two methods.

In the following only results obtained by the taboo search are reported as the taboo search and genetic algorithm appear equivalent for optimizing the permutation of jobs.

VI. NUMERICAL RESULTS

This section tests the efficiency of our approach on a large suite of benchmarks. These benchmarks are composed of three collections of instances depending on the presence of the multiresource and the flexibility features. The heuristic was coded in C language and ran on personal computer with 2.8 GHz processor and 524 Mb RAM. In the remainder, the acronym MX denotes our heuristic.

A. Job-Shop With Blocking

Two sets of instances have been used to evaluate the quality of the heuristic. The first set is used to compare our heuristic with optimal solutions given by [16] and with the best solutions generated by the dedicated metaheuristic proposed in [22]. The second set consists of new medium-size instances that could be used as benchmarks in the future to compare the solution approaches. The first set contain the eighteen 10 jobs $\times$ 10 machines instances used in [16] and [22], where the instances of the classical job-shop were generalized by introducing the blocking feature. Table II presents the results. Column C_{opt} gives the optimum makespan, which was computed with a branch and bound algorithm in [16]. The best makespan obtained by the rollout metaheuristic proposed in [22] is presented in column $C_{rollout}$. This metaheuristic takes in average 50 s to obtain the best makespan on a PC with a Pentium II 350 MHz

TABLE II
COMPARISON WITH EXISTING APPROACHES ON 10×10 INSTANCES

Data	C_{opt}	$C_{rollout}$	MX(0)	MX(5)		MX(60)		MX(200)	
				C_{mean}	C_{best}	C_{mean}	C_{best}	C_{mean}	C_{best}
Abz5	1641	1838	1960	1805	1774	1744	1669	1705	*1641
Abz6	1249	1419	1564	1380	1356	1314	1298	1300	1298
Mt10	1158	1447	1381	1265	1233	1242	1227	1231	1227
Orb1	1256	1335	1448	1365	1344	1323	1295	1300	*1256
Orb2	1144	1370	1375	1257	1227	1222	1204	1208	1181
Orb3	1311	∞	1382	1352	1329	1328	1313	1316	*1311
Orb4	1246	1460	1484	1354	1294	1298	1288	1289	1288
Orb5	1203	1414	1444	1301	1251	1261	1243	1247	1243
Orb6	1266	1448	1504	1379	1315	1331	1306	1310	1301
Orb7	527	565	650	574	562	563	556	552	549
Orb8	1139	1223	1223	1183	1166	1160	*1139	1155	*1139
Orb9	1130	1329	1422	1248	1226	1175	1162	1165	1162
Orb10	1367	1789	1711	1485	1447	1468	1447	1432	1402
La16	1148	1231	1462	1277	1257	1235	1223	1221	1205
La17	968	1146	1252	1103	1020	1046	1020	1029	1020
La18	1077	1334	1385	1200	1156	1156	1156	1156	1156
La19	1102	1314	1392	1249	1228	1221	1210	1196	1191
La20	1118	1357	1362	1261	1207	1222	1204	1207	1204
MRE			21	9.5	6.4	6	4.5	4.7	3.6

processor and 128 MB of RAM. Due to the probabilistic nature of our heuristic, we run it ten times for each instance and consider the mean and best makespans. The makespan obtained on the initial sequence is given in column $MX(0)$. The makespan obtained after 60 and 200 s are given in column $MX(60)$ and column $MX(200)$, respectively. Since we used a computer with 2.8 GHz processor and 524 MB RAM; and in order to make a precise comparison with the metaheuristic of [22], we give the makespan obtained by our heuristic after only 5 s $(50 \times 350/2800 = 6.25)$. Finally, the mean relative errors $MRE(\%)$ indicate the deviation of the makespans from the optimum makespan.

Observe first the results given by $MX(5)$ and the ones of [22]. Out of 18 instances, the average (resp. best) values obtained by our heuristic is better in 15 (resp. 16) cases and for the three (resp. two) remaining instances for which our heuristic is outperformed the standard deviation is less than 2.5% (resp. 1.34%). This comparison confirms the efficiency of our approach on the particular problem of job-shop with blocking. We can see that the best metaheuristic proposed in [22] cannot find very good solutions for the job-shop problem with blocking (see for example the instance orb10 for which the deviation between our C_{best} and the one proposed in [22] is 23.6%). We can see also that the best metaheuristic sometime fails to obtain even a feasible solution (see instance orb3). However, our heuristic always generates feasible schedules.

Compare now the results given by C_{mean} (resp. C_{best}) and the optimal solutions. First the mean relative error of $MX(0)$ from the optimal values is about 21%. Our heuristic improves considerably the initial makespan, and continues to find better solutions. The mean relative error from the optimal values drops to 9.5% (resp. 6.4%) after 5 s, to 6% (resp. 4.5%) after 60 s and to 4.7% (resp. 3.6%) after 200 s. Furthermore, our approach is able to find four optimal solutions after 200 s.

Test problems La16-La20 seem more difficult to solve by our heuristic (and by the metaheuristic of [22]) than test problems Orb1-Orb10. The mean relative error is 6.7% for La16-La20, while it is 2.2% for test problems Orb1-Orb10. This might be caused by the structure of the test problems Orb1-Orb10 that tends to flow-shop instances. It seems that test problems get easier as the number of deadlock situation decreases.

The second set of instances is composed of 40 instances which introduce the blocking constraint to the classical job-shop instances of [15] that were extensively used for the classical job-shop problem. Notice that instances LA16-20 are already investigated in the first set of experiments. Table III shows the mean and best values of ten independent runs of MX each of which is performed with 200 s. The initial makespan $MX(0)$ is also reported.

To our knowledge, the optimal makespan (or even tight lower bounds) are not known for these instances. We notice that the heuristic seems more stable for instances with five machines. The value obtained by the ten replications is the same for 10×5 instances. The mean relative error between the mean makespan and the best one is about 2.5% for 15×5 and 20×5 instances. The mean relative error reaches about 4.3% for 15×10 instances.

B. Job-Shop With Multipurpose Machines and Blocking

The test problems in this sample are job-shops with multipurpose machines ($JMPM$) to which we add the blocking feature. Two types of comparison of different problem sizes were performed. For small size instances, the optimization engine XPRESS is used to solve the MIP formulation of Section II in order to evaluate our heuristic. Thirty test problems are generated from the *edata* instances of [13] that are characterized by a small number of flexibility (each operation is associated with less than two machines). The first (resp. last) 15 instances are 8×5 (resp. 6×10) instances derived from the instances of [13] by keeping only the first 8 (resp. 6) jobs. The motivation of addressing only test problems in *edata* is that the mathematical programming package we used is not able to obtain good solutions for instances with medium (*rdata*) and large number of flexibilities (*vdata*) of [13] due to the very poor quality of lower bounds. The effectiveness of the proposed heuristic was analyzed in terms of best makespan found by MX compared to the best value provided by solving the MIP formulation of

TABLE III
RESULTS ON MODIFIED LAWRENCE'S INSTANCES

Data	n	m	$MX(0)$	*MX*(200)		Data	n	m	$MX(0)$	*MX*(200)	
				C_{mean}	C_{best}					C_{mean}	C_{best}
La01	10	5	956	881	881	La21	15	10	2109	1771	1700
La02	10	5	1003	900	900	La22	15	10	1974	1617	1519
La03	10	5	927	808	808	La23	15	10	1944	1779	1731
La04	10	5	1043	862	862	La24	15	10	1887	1716	1633
La05	10	5	878	742	742	La25	15	10	2045	1702	1655
La06	15	5	1505	1281	1243	La26	20	10	2583	2353	2310
La07	15	5	1334	1209	1194	La27	20	10	2705	2428	2362
La08	15	5	1458	1261	1216	La28	20	10	2754	2381	2291
La09	15	5	1559	1380	1349	La29	20	10	2400	2256	2184
La10	15	5	1475	1300	1270	La30	20	10	2646	2405	2312
La11	20	5	1887	1762	1725	La31	30	10	3596	3489	3403
La12	20	5	1734	1546	1479	La32	30	10	4127	3787	3701
La13	20	5	1988	1724	1683	La33	30	10	3797	3460	3363
La14	20	5	2065	1775	1745	La34	30	10	3854	3470	3380
La15	20	5	2105	1788	1749	La35	30	10	4004	3527	3373
La16	10	10	1415	1221	1205	La36	15	15	2258	2058	2026
La17	10	10	1353	1029	1020	La37	15	15	2444	2229	2177
La18	10	10	1307	1156	1156	La38	15	15	2208	2008	1968
La19	10	10	1332	1196	1191	La39	15	15	2307	2046	1992
La20	10	10	1505	1207	1204	La40	15	15	2341	2034	1958

TABLE IV
COMPARISON WITH LOWER BOUNDS AND OPTIMAL VALUES ON *JMPM* INSTANCES

Data	n	m	Opt(LB,UB)	*MX*		Data	n	m	Opt(LB,UB)	*MX*	
				$C_{\max}$	RE					$C_{\max}$	RE
La01	8	5	(674 715)	715	6.1	La16	6	10	834	*834	0.0
La02	8	5	(705 720)	720	2.1	La17	6	10	693	748	7.9
La03	8	5	(589 666)	655	11.2	La18	6	10	661	*661	0.0
La04	8	5	(601 782)	715	18.9	La19	6	10	802	807	0.6
La05	8	5	(513 717)	613	19.5	La20	6	10	659	683	3.6
La06	8	5	(669 783)	719	7.4	La21	6	10	708	746	5.3
La07	8	5	(597 657)	646	8.2	La22	6	10	719	727	1.1
La08	8	5	(622 650)	687	10.4	La23	6	10	774	778	0.5
La09	8	5	(686 764)	762	11.0	La24	6	10	694	700	0.8
La10	8	5	(660 819)	709	7.4	La25	6	10	725	782	7.8
La11	8	5	(599 940)	715	19.3	La26	6	10	794	827	4.1
La12	8	5	(545 744)	676	24.0	La27	6	10	664	731	10.1
La13	8	5	(542 640)	643	18.6	La28	6	10	691	707	2.3
La14	8	5	(644 708)	746	15.8	La29	6	10	776	808	4.1
La15	8	5	(594 664)	670	12.8	La30	6	10	(790 826)	809	2.4
MRE					12.8						3.3

Section II with $M = \sum_{i=1}^{n} \sum_{j=1}^{n_i} p_{i,j}$ and $\epsilon = 0.01$. If the optimal solution is not found after 10 h and after exploring more than 8 millions of nodes in the search tree, the mathematical programming package stops with a lower bound and an upper bound. Notice that, due to the small number of jobs, the results of MX are obtained by evaluating all job sequences which is achieved in less than 3 s of CPU time.

Table IV shows the results of our approach as well as optimal makespan (*Opt*), lower bound (*LB*), and upper bound (*UB*) obtained by the mathematical programming package on the 30 test problems. Column MX gives respectively the best makespan ($C_{\max}$) of our approach and the relative errors [*RE*(%)] that are evaluated as follows $100 \times (C_{\max} - Opt)/Opt$ or $100 \times (C_{\max} - LB)/LB$ depending on whether the optimal makespan is available.

The test problems with eight jobs and five machines are more difficult to solve than the test problems with six jobs and ten machines. The mathematical programming package could not obtain the optimal value for the 15 instances. Consider the quality of our approach with respect to the best solutions obtained by the mathematical programming package. Out of 15 test problems, our approach is better in nine cases, similar in two cases, whereas it is outperformed in the remaining four cases with an average deviation of 3%. The relative errors from the lower bounds are quite large for this first set of test problems. This is most likely caused by poor lower bounds generated by the mathematical programming package.

For the test problems with six jobs and ten machines, all optimal solutions are obtained by the mathematical programming package except for instance La30. Instances La26-La30 seem to be more difficult to solve than the other instances. The computation time needed to prove the optimality is on average greater than 1/2 h. Concerning the results obtained by our approach, even though the optimal value is only obtained for 2 over 15 instances, the results are rather satisfactory. The mean relative error from the optimal makespan is equal to 3.3%. For test problem La30, the makespan of our approach is better than the best solution obtained by the mathematical programming package. Note that MX needs less than 30 s to evaluate all job sequences.

The second type of comparison is carried out on medium size instances with 10, 15, and 20 jobs. The number of machines is equal to 5 or 10. The test problems are instances generated in [13] for the job-shop with multipurpose machines. As men-

TABLE V
RESULTS ON MEDIUM SIZE *JMPM* INSTANCES

Data	n	m	*edata*			*rdata*			*vdata*		
			MX(0)	MX_{mean}	MX_{best}	MX(0)	MX_{mean}	MX_{best}	MX(0)	MX_{mean}	MX_{best}
La01	10	5	1203	905	895	1291	817	799	1030	758	748
La02	10	5	1149	852	852	1056	728	727	975	688	672
La03	10	5	1195	760	757	1044	667	660	935	646	641
La04	10	5	1122	826	823	1015	721	705	990	680	677
La05	10	5	1120	717	717	993	661	655	778	625	623
La06	15	5	1669	1281	1259	1654	1193	1178	1532	1121	1079
La07	15	5	1663	1192	1140	1332	1102	1079	1495	1047	1018
La08	15	5	1796	1259	1231	1499	1122	1110	1476	1066	1026
La09	15	5	1757	1377	1349	1517	1241	1224	1543	1164	1157
La10	15	5	1693	1273	1247	1428	1181	1164	1478	1084	1068
La11	20	5	2114	1730	1689	2030	1635	1619	2240	1518	1497
La12	20	5	2120	1561	1526	1712	1416	1398	1787	1355	1322
La13	20	5	1993	1690	1673	2020	1541	1507	2013	1510	1466
La14	20	5	2223	1737	1700	2065	1607	1572	1990	1520	1509
La15	20	5	2272	1763	1745	1981	1620	1590	1888	1579	1533
La16	10	10	1678	1200	1198	1585	1100	1093	1562	963	919
La17	10	10	1255	1026	1002	1200	948	920	1450	841	825
La18	10	10	1418	1135	1127	1681	1078	1070	1203	947	897
La19	10	10	1397	1172	1155	1493	1099	1081	1343	1024	1001
La20	10	10	1700	1151	1147	1688	1124	1114	1199	981	930
La21	15	10	2542	1767	1737	2139	1582	1540	1758	1342	1291
La22	15	10	1873	1585	1557	1836	1475	1453	2621	1215	1173
La23	15	10	2321	1729	1692	2266	1627	1602	2084	1333	1276
La24	15	10	2173	1667	1642	1682	1536	1474	2554	1325	1245
La25	15	10	2006	1642	1621	2144	1499	1465	1594	1287	1249
La26	20	10	2533	2302	2274	2704	2150	2123	2568	1966	1917
La27	20	10	2953	2357	2327	3169	2203	2088	2505	2052	2008
La28	20	10	2554	2311	2279	3248	2191	2147	2472	2013	1960
La29	20	10	2403	2212	2192	2735	2064	2022	2450	1845	1781
La30	20	10	2928	2317	2230	3105	2203	2163	2835	2070	2009

TABLE VI
RESULTS ON MULTIMODE INSTANCES

Data	n	m	*emdata*		*erdata*		*evdata*	
			MX(0)	MX_{mean}	MX(0)	MX_{mean}	MX(0)	MX_{mean}
la01	10	5	1960	1515	2454	2197	2537	2250
la02	10	5	1698	1341	1995	1672	2531	1986
la03	10	5	1715	1362	1891	1749	2057	1843
la04	10	5	1680	1387	2131	1661	2270	2024
la05	10	5	1672	1064	1978	1672	2117	1857
la16	10	10	5067	4518	3645	2929	5279	5112
la17	10	10	4570	3979	3227	2449	4676	4661
la18	10	10	4901	4570	3293	2855	5128	5001
la19	10	10	4880	4450	3230	2784	5239	5060
la20	10	10	4996	4602	3287	2793	5445	5401

tioned before, three sets of test problems with different degree of flexibilities were generated. Each operation is associated on average with $m/2$ machines in a *vdata* instance of high flexibility, with two machines in an *rdata* instance of medium flexibility, and with less than two machines in an *edata* instance of little flexibility. To our knowledge, the optimal makespan (or even tight lower bounds) are not known for these instances. Note that tight lower bounds are available for these instances when the blocking feature is not considered, they become very poor if the blocking feature is added.

In order to analyze the mean makespan with regard to the best makespan, ten independent runs of MX of 200 s each, are carried out. Table V shows the results in which column $MX(0)$, MX_{mean}, and MX_{best} give, respectively, the initial, the mean, and the best makespan.

Again, considerable improvements are achieved by optimizing the job sequence using taboo search. This implies that even though the greedy algorithm is good, its performance depends heavily on the job sequence it uses. Comparison of Columns MX_{mean} and MX_{best} shows that the proposed heuristic MX does not generate poor solutions from one run to another. For instance, the mean relative error between the mean makespan and the best one is about 0.7% for 10×5 instances. The mean relative error increases with the size of the problem and varies from *edata*, *rdata* and *vdata* instances.

C. Multimode Job-Shop With Blocking

The last sample in our experiment is dedicated to the multimode job-shops with blocking. It is derived from 30 instances of [2] with ten jobs, which are randomly generated from the *edata* instances of [13]. Instances La01-La05 contain five resources, while there are ten resources in instances La16-La20. Again, ten independent runs of MX are carried out with 200 s for each of them. Table VI shows the results, in which column $MX(0)$ gives the makespan of the initial solution and column MX_{mean} gives the mean makespan of the ten runs.

Due to lack of tight lower bounds for the multimode job-shop with blocking, the results presented in Table VI cannot be evaluated. However, it can be seen, especially for *emdata* and *erdata* instances, that the mean makespan after 200 s considerably improves the initial makespan. This confirms the need of the taboo search procedure for optimizing the job sequence. However, for *evdata* instance, the initial makespan seems good comparing with the mean makespan. For instance, the relative error is less

than 1% for instance La20. The reason might come from the fact that the test problems get easier as the flexibility increases since the number of blocking and deadlock situations decreases.

VII. Conclusion

This paper has addressed a very general shop scheduling that captures main features of modern production systems including multiresource operations, resource flexibilities of operations and the blocking constraint for resource release. The blocking constraint requires to hold resources used for an operation till resources needed for the next operation of the same job are available. A shortest path approach extending the classical geometric approach is proposed for the two-job case. A greedy heuristic is then proposed to schedule several jobs by considering the jobs sequentially according to a job sequence, grouping scheduled jobs into a combined job and then scheduling the combined job and the next unscheduled job with the shortest path approach. A metaheuristic is then used to identify effective job sequences. Extensive numerical experimentation proves the efficiency of our approach.

It is worth mentioning the basic approach proposed in this paper has been used to efficiently solve other classes of scheduling problems. Future research consists in extending the methodology to other classes of scheduling problems with resources available in multiple units, operations having resource flexibilities but with processing times depending on the choice of resource assignment, jobs with multiple manufacturing processes. Another class of problems under consideration concerns no-wait scheduling. Further, more research is needed to define good quality lower bounds.

Appendix

Progress in the $2D$-Plane Representation of the Two Flexible Jobs: We only consider N_2 and the progress for nodes of N_1 is identical. Starting from a node $v_0 = (k_1^0, k_2^0, *, *)$, the progress identifies all nodes that can be reached by first progressing diagonally then horizontally/vertically. The concept of modified sets of candidates, denoted by AM_{i,k_i}, is used to record the alternative subsets that remain feasible for an operation O_{i,k_i} as we progress in the $2D$-plane. As the progress depends on the processing times of operations, the diagonal $DIAG$ can hit either an upper boundary, a right boundary, or a NE corner of a rectangle (k_1, k_2).

If $DIAG$ hits the upper boundary, the next diagonal progress consists in progressing job J_2 to O_{2,k_2+1}, while holding all resources of O_{1,k_1}. Thus, feasible alternative subsets of AM_{2,k_2+1} of O_{2,k_2+1} are the ones that do not share any resource with at least one feasible subset of AM_{1,k_1} of O_{1,k_1}. After AM_{2,k_2+1} is updated, some alternatives of O_{1,k_1} becomes infeasible, and AM_{1,k_1} must be modified by eliminating alternatives in resource conflict with all alternatives of AM_{2,k_2+1}.

Similarly, the determination of the new alternative subsets if $DIAG$ hits a right boundary of a rectangle, is performed by first calculating the new subsets AM_{1,k_1+1}, and then updating AM_{2,k_2}.

Finally, if the diagonal $DIAG$ hits the NE corner of a rectangle, the next diagonal progress highly depends on the job that progresses first to its next operation. If J_1 (resp. J_2) progresses first, we first determine AM_{1,k_1+1} (resp. AM_{2,k_2+1}) and then AM_{2,k_2+1} (resp. AM_{1,k_1+1}). The final modified candidate sets AM_{1,k_1+1} and AM_{2,k_2+1} are unions of the related sets determined in both scenarios.

The determination of the immediate successors of a SE corner v_0 is summarized in Algorithm A3. In the case where the progress starts from a NW corner $v_0 = NW(k_1^0, k_2^0, A^r_{1,k_1^0-1}, *)$, the only modification to introduce to Algorithm A3 concerns the first step. This step becomes $k_1 \leftarrow k_1^0, k_2 \leftarrow k_2^0+1$ and the vertical progress is used with k_1, k_2 and A^r_{1,k_1-1}. In the case where v_0 corresponds to the origin, the first step is $k_1 \leftarrow 1, k_2 \leftarrow 1$ and both horizontal and vertical progresses are performed.

Algorithm A3. Progress From $v_0 = SE(k_1^0, k_2^0, *, a^s_{2,k_2^0-1})$:

Step 1: Set $k_1 \leftarrow k_1^0+1, k_2 \leftarrow k_2^0$ make a *horizontal progress* with k_1, k_2 and A^s_{2,k_2-1} in order to reach the first SE corner of a potential obstacle.

Step 2: Progress diagonally till hitting a rectangle boundary. Update the alternative subsets according to the boundary crossed.

Step 3: If AM_{1,k_i} or AM_{2,k_i} becomes empty, then stop.

Step 4: If $DIAG$ hits the upper boundary of a rectangle (k_1, k_2) then make a *horizontal progress* with k_1, k_2 and AM_{2,k_2} in order to reach the first SE corner of a potential obstacle.

Step 5: If $DIAG$ hits the right boundary of a rectangle (k_1, k_2) then make *vertical progress* with k_1, k_2, and AM_{1,k_1} in order to reach the first NW corner of a potential obstacle.

Step 6: Go to Step 2.

The procedure *horizontal progress* used in Steps 1 and 4 explores the horizontal line k_2 in order to determine the SE corner of the first potential obstacle. If this obstacle is found, the progress is stopped and nodes of the corresponding SE corner, with all the alternative subsets that allow reaching it, are considered as immediate successors of v_0. The horizontal progress eliminates progressively the candidate subsets of the related operation of J_2. The progress stops if no alternative subset of $O_{1,k_1'}$ allows feasible horizontal progress, or the right boundary of the final obstacle is reached ($k_1' \geq n_1$).

Similarly, in Step 5 the procedure *vertical progress* determines the NW corner of the first potential obstacle reached on the vertical k_1. Formal description of the horizontal and vertical progresses is neglected for the sake of space limitation.

Resource Assignment Along a Given Path P: Note that an optimal path P on the $2D$-plane can be decomposed into subpath $SP(v_0, v_e)$ connecting nodes v_0 and v_e of N_2 by progressing first diagonally then horizontally/vertically. A partial assignment which enables us to get from v_0 to v_e may be constructed from the modified alternative subsets constructed during the exploration. The basic idea for the assignment of alternative subsets is to: (i) apply Algorithm A3 by restricting to progress along $SP(v_0, v_e)$ to determine sets $AM_{i,j}$ and (ii) backtracking the subpath $SP(v_0, v_e)$ by selecting resources according to the modified set $AM_{i,j}$. The details of the progress are given in Algorithm A6 for the case $v_e = SE(k_1^e, k_2^e, *, A^s_{2,k_2-1})$. The

case $v_e = NW(k_1^e, k_2^e, A^r_{1,k_1-1}, *)$ is similar. The notation $FEAS(F, A) = \{A' \in F \mid A' \cap A = \emptyset\}$ is used.

Algorithm A4. Resource Assignment Along $SP(v_0, V_e)$:

Step 1: Apply Algorithm A3 for v_0 by restricting the progress along $SP(v_0, v_e)$. Let $k_1 \leftarrow k_1^e, k_2 \leftarrow k_2^e - 1$.

Step 2: While O_{2,k_2^e} is totally covered by the horizontal segment of SP:
— select $A^r_{1,k_1} \in FEAS(F_{1,k_1}, A^s_{2,k_2})$;
— $k_1 \leftarrow k_1 - 1$.

Step 3: Select $A^r_{1,k_1} \in FEAS(AM_{1,k_1}, A^s_{2,k_2})$.

Step 4: Backtrack the subpath along the diagonal till hitting a new boundary of a rectangle. If v_0 is reached, then stop.

Step 5: If the diagonal hits a new upper boundary then:
— $k_2 \leftarrow k_2 - 1$;
— select $A^s_{2,k_2} \in FEAS(AM_{2,k_2}, A^r_{1,k_1})$.

Step 6: If the diagonal hits a new right boundary then:
— $k_1 \leftarrow k_1 - 1$;
— select $A^r_{1,k_1} \in FEAS(AM_{1,k_1}, A^s_{2,k_2})$.

Step 7: Go to Step 4

REFERENCES

[1] C. Arbib, G. Italiano, and A. Panconesi, "Predicting deadlock in store-and-forward networks," *Networks*, vol. 20, pp. 861–881, 1990.

[2] P. Brucker and J. Neyer, "Tabu search for the multi-mode job-shop problem," *Operations Research Spektrum*, vol. 20, pp. 21–28, 1998.

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

[4] V. Caraffa, S. Ianes, T. Bagchi, and C. Sriskandarajah, "Minimizing makespan in a blocking flowshop using genetic algorithms," *Int. J. Prod. Econ.*, vol. 70, pp. 101–115, 2001.

[5] A. D'Ariano, D. Pacciarelli, and M. Pranzo, "A branch and bound algorithm for scheduling trains in a railway network," *Eur. J. Oper. Res.*, vol. 183, pp. 643–657, 2007.

[6] B. Damasceno and X. Xie, "Scheduling and deadlock avoidance of a flexible manufacturing system," in *Proc. IEEE Conf. Syst., Man, Cybern.*, San Francisco, CA, 1998, vol. 1, pp. 564–569.

[7] B. Damasceno and X. Xie, "Petri nets and deadlock-free scheduling of multiple-resource operations," in *Proc. IEEE Conf. Syst., Man, Cybern.*, Tokyo, Japan, 1999, vol. 1, pp. 878–883.

[8] S. Dauzère-Pérès and J. Paulli, "An integrated approach for modeling and solving the multiprocessor job-shop scheduling problem using tabu search," *Ann. Oper. Res.*, vol. 70, pp. 281–306, 1997.

[9] S. Dauzère-Pérès, W. Roux, and J. Lasserre, "Multi-resource shop scheduling with resource flexibility," *Eur. J. Oper. Res.*, vol. 107, pp. 289–305, 1998.

[10] M. Drozdowski, "Scheduling multiprocessor tasks—an overview," *Eur. J. Oper. Res.*, vol. 94, pp. 215–230, 1996.

[11] J. Grabowski and J. Pempera, "Sequencing of jobs in some production system," *Eur. J. Oper. Res.*, vol. 125, pp. 535–550, 2000.

[12] N. Hall and S. Sriskandarajah, "A survey of machine scheduling problems with blocking and no-wait in process," *Oper. Res.*, vol. 44, no. 3, pp. 510–525, 1996.

[13] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines," *Operations Research Spektrum*, vol. 15, pp. 205–215, 1994.

[14] A. Kumar, Prakash, M. Tiwari, R. Shankar, and A. Baveja, "Solving machine-loading problem of a flexible manufacturing system with constraint-based genetic algorithm," *Eur. J. Oper. Res.*, vol. 175, pp. 1043–1069, 2006.

[15] S. Lawrence, Supplement to resource constrained project scheduling: An Experimental investigation of heuristic scheduling techniques Carnegie-Mellon Univ., Pittsburgh, PA, 1984.

[16] A. Mascis and D. Pacciarelli, "Job-shop scheduling with blocking and no-wait constraints," *Eur. J. Oper. Res.*, vol. 143, pp. 498–517, 2002.

[17] Y. Mati, "Scheduling problems in automated manufacturing systems : Models, complexities and solution methods," Ph.D. dissertation, University of Metz, Metz, France, 2002.

[18] Y. Mati, N. Rezg, and X. Xie, "Geometric approach and taboo search for scheduling flexible manufacturing systems," *IEEE Trans. Robot. Autom.*, vol. 17, pp. 805–818, 2001.

[19] Y. Mati, N. Rezg, and X. Xie, "A taboo search approach for deadlock-free scheduling of automated manufacturing systems," *J. Intell. Manuf., Special Issue on Metaheuristics*, vol. 12, no. 5/6, pp. 535–552, 2001.

[20] Y. Mati and X. Xie, "The complexity of the two-job shop scheduling problems with multi-purpose unrelated machines," *Eur. J. Oper. Res.*, vol. 152, pp. 159–169, 2004.

[21] Y. Mati and X. Xie, "A genetic search guided greedy algorithm for multi-resource shop scheduling with resource flexibility," *IIE Trans.*, vol. 40, pp. 1228–1240, 2008.

[22] C. Meloni, D. Pacciarelli, and M. Pranzo, "A rollout metaheuristic for job shop scheduling problems," *Ann. Oper. Res.*, vol. 131, pp. 215–235, 2004.

[23] I. M. Oliver, D. J. Smith, and J. Holland, "A study of permutation crossover operators on the traveling salesman problem," Texas Instruments Ltd., Dallas, TX, 1984.

[24] D. Pacciarelli and M. Pranzo, "Production scheduling in a steel-making-continuous casting plant," *Comput. Chem. Eng.*, vol. 28, pp. 2823–2835, 2004.

[25] J. Paulli, "A hierarchical approach for the FMS scheduling problem," *Eur. J. Oper. Res.*, vol. 86, pp. 32–42, 1995.

[26] S. Ramaswamy and S. Joshi, "Deadlock-free schedules for automated manufacturing workstations," *IEEE Trans. Robot. Autom.*, vol. 12, pp. 391–400, 1996.

[27] H. K. Roh and Y. D. Kim, "Due-date based loading and scheduling methods for a flexible manufacturing system with an automatic tool transporter," *Int. J. Prod. Res.*, vol. 35, pp. 2989–3004, 1997.

[28] D. P. Ronconi, "A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking," *Ann. Oper. Res.*, vol. 138, no. 1, pp. 53–65, 2005.

[29] T. Sawik, "Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers," *Mat. Comput. Modelling*, vol. 31, no. 13, pp. 39–52, 2000.

[30] M. K. Tiwari, S. Kumar, and R. Shankar, "Solving part-type selection and operation allocation problems in an FMS: An approach using constraints-based fast simulated annealing algorithm," *IEEE Trans. Syst., Man, Cybern.*, vol. 36, pt. A, pp. 1170–1184, 2006.

[31] X. Wang and L. Tang, "A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers," *Comput. Oper. Res.*, vol. 36, pp. 907–918, 2009.

[32] H. Xiong and M. Zhou, "Scheduling of semiconductor test facility Perti nets and hybrid heuristic search," *IEEE Trans. Semicond. Manuf.*, vol. 11, pp. 384–393, 1998.

Yazid Mati born in Emir AEK, Jijel, Algeria, in 1975. He received the Diploma degree in operations research from the Université des Sciences et Technologies Houari Boumediene, Alger, Algeria, in 1997, and the DEA diploma in industrial engineering from the Institut National Polytechnique de Grenoble, Grenoble, France, in 1999, and the Ph.D. degree in industrial engineering from the Université de Metz, Metz, France, in 2002.

He was an Assistant Professor at the Ecole des Mines de Nantes, France, from 2002 to 2004, an Associate Professor at the Ecole Nationale Supérieure des Mines de Saint Etienne (ENSM.SE), a 190-year old top-10 French graduate engineering school, from 2004 to 2005. In September 2005, he joined the Al-Qassim University, Kingdom of Saudi Arabia, where he is currently an Associate Professor of Industrial Engineering. His current research interests include production planning and scheduling and supply chain optimization.

Xiaolan Xie received the Ph.D. degree from the University of Nancy I, Nancy, France, in 1989, and the Habilitationà Diriger des Recherches degree from the University of Metz, Metz, France, in 1995.

Currently, he is a Full Professor in Industrial Engineering and Head of the Department of Healthcare Engineering, Ecole Nationale Supérieure des Mines de Saint Etienne (ENSM.SE), a 190-year old top-10 French graduate engineering school. Before joining ENSM.SE, he was a Research Director at the Institut National de Recherche en Informatique et en

Automatique (INRIA) from September 2002 to March 2005, a Full Professor at the Ecole Nationale d'Ingénieurs de Metz (ENIM), a French engineering school from 1999 to 2002, and a Senior Research Scientist at INRIA from 1990 to 1999. His research interests include design, planning and scheduling, supply chain optimization, performance evaluation, and maintenance of manufacturing and healthcare systems. He is author/coauthor of over 200 publications including over 80 journal articles in leading journals of related field and five books. He has rich industrial application experiences with European industries. He is the French-Leader for the European Project FP6-IST6 IWARD on swarm robots for health services, for FP6-NoE I*PROMS on intelligent machines and production systems, for the GROWTH-ONE project for the strategic design of supply chain networks, and for the GROWTH thematic network TNEE on extended enterprises.

Dr. Xie is an Associate Editor on the Conference Editorial Board of IEEE Robotics and Automation Society and was an Associate Editor of IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING, the IEEE TRANSACTIONS ON AUTOMATIC CONTROL, and the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION (2001–2004). He has served as a Guest Editor of special issues and special volumes in various journals such as *Annals of Operations Research*, *Health Care Management Science*, the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, *the International Journal of Production Research*, IJCIM. He was General Chair for ORAHS'2007 and IPC Chair for IEEE Workshop Health Care Management (WHCM 2010), and has been served on International Program Committees for many conferences.