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 require 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 require 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. 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 $\mathcal{J} = \{J_1, J_2, \ldots, J_N\}$ of $N$ jobs must be performed on a set $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$ 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 $\{O_{i1}, O_{i2}, \ldots, O_{in_i}\}$ 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 $O_{ij}$ needs simultaneously a subset of resources $\mathcal{A}_{ij} \subseteq \mathcal{R}$. Thus, two operations $O_{rs}$ and $O_{pq}$ cannot be processed concurrently if they share common resources, i.e., $\mathcal{A}_{rs} \cap \mathcal{A}_{pq} \neq \emptyset$.
- **Multimode property.** The subset of resources $\mathcal{A}_{ij}$ to be used to perform an operation $O_{ij}$ can be selected in a set $F_{ij} = \{A_{ij1}, A_{ij2}, \ldots, A_{ijm_{ij}}\}$ of $m_{ij}$ alternative subsets also called candidate subsets. The processing time $p_{ij}$ of operation $O_{ij}$ 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 $F_{ij}$ of alternative subsets contains a singleton for all operation $O_{ij}$ 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 $O_{ij}$: i) all resources used for $O_{ij}$ are held until all resources needed for the next operation $O_{ij+1}$ of the same job become available and ii) then resources used for $O_{ij}$ but not needed for $O_{ij+1}$ are released and new resources needed for $O_{ij+1}$ 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 $O_{ij}$, then its resource requirement can be simply represented by $F_{ij} = \{M(p_{ij})\}$, where $M$ is the related machine for $O_{ij}$ and $p_{ij}$ the processing time. If we want to further take into account human resources and tools, then $F_{ij} = \{M \cap OP \cap T(p_{ij})\}$, where $OP$ is the competent operator for operation $O_{ij}$ and $T$ the machining tool needed for $O_{ij}$. 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 $O_{ij1}$ and $O_{ij2}$ with $F_{ij1} = \{M \cap OP \cap T(\Delta_{ij})\}$ and $F_{ij2} = \{M \cap T(p_{ij})\}$, where $\Delta_{ij}$ 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 $F_{ij} = \{\emptyset(0)\}$, waiting in a buffer of size $n$ as $F_{ij} = \{B_1 \cup B_2 \cup \cdots \cup B_n(0)\}$, where $B_i$ denotes the $i$th storage unit of the buffer, transportation delay $\Delta$ as $F_{ij} = \{\emptyset(\Delta)\}$, transportation with a robot $R$ as
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 defining the alternative ways of performing an operation. To see this, let \( O_{i,j} \) be an operation that needs two resources \( R_j \) 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 \( \{R_j, R_{2,1}\} \), or \( \{R_j, R_{2,2}\} \) or \( \{R_j, R_{2,3}\} \).

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 \( k \), \( X_{ij,pq} \) is equal to 1 if operation \( O_{i,j} \) is sequenced before \( O_{p,q} \) and \( S_{k,i,j} \) 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 \( M \geq 0 \) 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 \( O_{i,j} \), 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 \( \epsilon \) 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 \( S_{1,1,i} = 0, S_{1,2,i} = 1, S_{2,1,i} = 0, S_{2,2,i} = 1 \) 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.

\[
\text{Minimize} \quad C_{\text{max}} \tag{1}
\]

\[
\begin{align*}
S_{i,j} + p_{i,j} \geq & \quad S_{i,j} \quad \forall i \in \mathbb{S}, \quad \forall j = 1, \ldots, n_i - 1 \\
\sum_{k=1}^{n_i} Y_{ij}^k = & \quad 1 \quad \forall i \in \mathbb{S}, \quad \forall j = 1, \ldots, n_i \\
S_{i,j} - S_{p,q} + 1 \geq & \quad \epsilon - M \left( 2 - Y_{ij}^k - Y_{pq}^k + X_{ij,pq} \right) \quad \forall O_{i,j}, O_{p,q}, k, k' | A_{ij}^k \cap A_{pq}^k \neq \emptyset \tag{3}
\end{align*}
\]

\[
\begin{align*}
S_{p,q} - S_{i,j} + 1 \geq & \quad \epsilon - M \left( 3 - Y_{ij}^k - Y_{pq}^k - X_{ij,pq} \right) \quad \forall O_{i,j}, O_{p,q}, k, k' | A_{ij}^k \cap A_{pq}^r \neq \emptyset \tag{4}
\end{align*}
\]

\[
C_{\text{max}} \geq S_{i,j} + p_{i,j} \quad \forall i \in \mathbb{S} \tag{5}
\]

\[
X_{ij,pq} \in \{0, 1\}, \quad S_{i,j} \geq 0
\]
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 $R_3$ for two time units, then $R_4$ for one time unit, and finally both $R_2$ and $R_4$ for four time units. Notation $J_2 = \{R_3(2)\} \{R_4(1)\} \{R_2R_4(4)\}$ is used. The first job is defined as $J_1 = \{R_2(4)\} \{R_3(2)\} \{R_3(2)\}$.

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 $O_{k,i,j}$ of length $p_{i,j}$ 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 $O_{k,1,i,j}$ and $O_{k,2,i,j}$ 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 $O_{k,1,i,j}$ of $J_1$ is in progress and $J_2$ has finished $O_{k,2,i,j}$, 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 $(k_3, k_2)$ 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
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\((k_1, k_2)\)
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 \(v_{ij}\), all nodes that can be reached by first going
diagonally and then vertically (resp. horizontally), are consid-
ered as successors of \(v_{ij}\) (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 neces-
sary 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 \(y'\) is obtained from \(v\) by first going diagonally and then hori-
izontally (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 corre-
sponding 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 \(J_2\).

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 con-
ists 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 re-
source-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 \(SP(c_{li}, c_{li+1})\), 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 algo-
rithm 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 \(A'_{i,j}\), is chosen for
each operation \(O_{i,j}\), a 2D-representation can be given, a net-
work can be constructed and the optimal schedule can be deter-
mined 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 assign-
ment and its optimal schedule can be determined from the net-
work. The remaining part of this subsection is organized as fol-
loows: (A) a 2D-plane representation; (B) construction of the net-
work; (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 $A^r_{k_1, k_1}$ and $A^s_{k_2, k_2}$ that share common resources, i.e., $A^r_{k_1, k_1} \in F^r_{k_1, k_1}$ and $A^s_{k_2, k_2} \in F^s_{k_2, k_2}$. $A^r_{k_1, k_1} \cap A^s_{k_2, k_2} = \emptyset$. It is considered as a real obstacle if $A^r_{k_1, k_1} \cap A^s_{k_2, k_2} = \emptyset$. $V A^r_{k_1, k_1} \cap A^s_{k_2, k_2}$.

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 $O$, $F$, $SE(k_1, k_2, A^r_{k_1, k_1}, A^s_{k_2, k_2})$ for all potential obstacles $(k_1, k_2)$ and for all alternative sets $A^r_{k_1, k_1}$ and $A^s_{k_2, k_2}$. Subsets $A^r_{k_1, k_1}$ and $A^s_{k_2, k_2}$ (resp. $A^r_{k_2, k_2}$ and $A^s_{k_1, k_1}$) givie 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 $v = SE(k_1, k_2, A^r_{k_1, k_1}, A^s_{k_2, k_2})$ with $(k_1, k_2)$ and $k_1 < n_1$ (resp. $v = NW(k_1, k_2, A^r_{k_2, k_2}, A^s_{k_1, k_1})$ 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 $v = SE(k_1, k_2, A^r_{k_1, k_1}, A^s_{k_2, k_2})$ (resp. $v = NW(k_1, k_2, A^r_{k_2, k_2}, A^s_{k_1, k_1})$). Details of the progress on the 2D-plane will be explained in the Appendix.

The length $d(v, v')$ 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 $N_0$ 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 presentation 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 $\zeta$ with $A^{(i, j)}_{k_1, k_2}$ 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 $\zeta$. By construction, if $s_i$ is a SE (resp. NW) corner of an obstacle $(k_1, k_2)$, then it is a node $SE(k_1, k_2, A^r_{k_1, k_1}, A^s_{k_2, k_2})$ (resp. $NW(k_1, k_2, A^r_{k_2, k_2}, A^s_{k_1, k_1})$) 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 $\xi$. 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 $SE(k_1, k_2, A^{(r, s)}_{k_1, k_1}, A^{(r, s)}_{k_2, k_2})$ or $NW(k_1, k_2, A^{(r, s)}_{k_2, k_2}, A^{(r, s)}_{k_1, k_1})$ of the 2D-flexible representation. Without loss of generality, let $c_1 = SE(k_1, k_2, A^{(r, s)}_{k_1, k_1}, A^{(r, s)}_{k_2, k_2})$ and $c_{i+1} = SE(k_1, k_2, A^{(r, s)}_{k_1, k_1}, A^{(r, s)}_{k_2, k_2})$. By construction, $SP(c_{i+1})$ is a path connecting $c_i$ to $c_{i+1}$ by starting with alternative subsets $(A^{(r, s)}_{k_1, k_1}, A^{(r, s)}_{k_2, k_2})$ at $c_i$, ending with alternative subsets $(A^{(r, s)}_{k_1, k_1}, A^{(r, s)}_{k_2, k_2})$ 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 $\xi$ 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 $v = SE(k_1, k_2, A^{r}_{k_1, k_1}, A^{s}_{k_2, k_2})$, 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 $O_{2,k_2}$ still holds resource subset $A^{s}_{k_2, k_2}$. Any alternative set $A^{r}_{k_1, k_1}$ such that $A^{r}_{k_1, k_1} \cap A^{s}_{k_2, k_2} = \emptyset$ 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 $A^{s}_{k_2, k_2}$ but is independent of $A^{r}_{k_1, k_1}$. After the start of operation $O_{1,k_1+1}$, operation $O_{2,k_2}$ might start as well and the alternative subset $A^{r}_{k_1, k_1}$ to be selected depends on the alternative subset selected for $O_{1,k_1+1}$. To summarize, possible evolution and selection beyond the corner $SE(k_1, k_2, A^{r}_{k_1, k_1}, A^{s}_{k_2, k_2})$ is totally independent of $A^{r}_{k_1, k_1}$. We then neglect this information in the definition of $SE$ corners and represent them as $SE(k_1, k_2, A^{s}_{k_2, k_2})$. Similarly, $NW$ corners can be represented as $NW(k_1, k_2, A^{r}_{k_1, k_1}, A^{s}_{k_2, k_2})$.

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 $O$, $F$, $SE(k_1, k_2, A^{s}_{k_2, k_2})$ and $NW(k_1, k_2, A^{r}_{k_1, k_1}, A^{s}_{k_2, k_2})$ 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 \textit{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 \(\text{REACHED} \leftarrow \{\text{\textit{O}}\}\).

**Step 2:** While \(\text{REACHED}\) is not empty:

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

- \(\text{REACHED} \leftarrow \text{REACHED} \cup \text{ADJACENT} \).

**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 \(O((T_1 + T_2)(T_1n_1 + T_2n_2) n_1n_2)\) time, where \(T_i = \max_{k=1,\ldots,m} |F_{i,k}|\).

**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 \((T_1n_1 + T_2n_2)\), where \(T_i = \max_{k=1,\ldots,m} |F_{i,k}|\). 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 \((T_1n_1 + T_2n_2)\). 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 \(O((T_1 + T_2)(T_1n_1 + T_2n_2) n_1n_2)\) time. Since the network is acyclic, Step 4 of the determination of the shortest path takes \(O(|V_2| + |V_2|)\). Consequently, the overall complexity of the Algorithm A1 is \(O((T_1 + T_2)(T_1n_1 + T_2n_2) n_1n_2)\).

**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, \(SE(k_1, k_2, *, *)\) and \(NW(k_1, k_2, A'_{i,k_1,i-1})\) as \(J_2\) is a fixed job. Our improvement concerns nodes \(NW(k_1, k_2, A'_{i,k_1,i-1})\). Note that, at \(NW(k_1, k_2, A'_{i,k_1,i-1})\), job \(J_2\) must progress before job \(J_1\). The information \(A'_{i,k_1,i-1}\) is necessary only to determine whether vertical progress can pursue. However, if the path chooses diagonal progress at \(NW(k_1, k_2, A'_{i,k_1,i-1})\), 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 \(FEAS(F;A) = \{A' \in F | A' \cap A = \emptyset\}\) 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 \(NW(k_1, k_2, A'_{i,k_1,i-1})\) is independent of \(A'_{i,k_1,i-1}\). What is important for the diagonal progress is to check the possibility of reaching a \(NW\) corner of type \(NW(k_1, k_2, A'_{i,k_1,i-1})\) 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 \(v_i = NW(k_1, k_2, *)\) 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 \(A'_{i,k_1,i-1}\) 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 \(\pi\) 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 \(\pi\), 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 \(\pi\). 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 \(\pi\). 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 \( J_{\text{com}} \) of the resulting schedule.
Step 4: For \( i = 3 \) to \( N \):
   - schedule \( J_{\text{com}} \) and \( J_{[i]} \) using the improved version of Algorithm A1 with \( J_{\text{com}} \) as a simple fixed job;
   - construct the new \( J_{\text{com}} \) of the resulting schedule.
Step 5: Derive the final schedule from \( J_{\text{com}} \).

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 \( O_{\text{com},j} \) 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 \( O_{k,j} \) of a job split in two consecutive operations \( O_{\text{com},k} \) and \( O_{\text{com},k+1} \) of the combined job does not release its resources at the end of \( O_{\text{com},k} \). Therefore, the nonpreemption of operations is satisfied.

Remark 5: The permutation \( \pi \) 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 \( \pi \).

Deriving a schedule from a combined job is straightforward by simply choosing for each operation \( O_{k,j} \) of a job the starting time of the corresponding combined operation in which operation \( O_{k,j} \) 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 \( O_{k,j} \) 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 \( J_{\text{com}}[i] \) be the combined job obtained at iteration \( i \). We prove by induction on \( i = 2, 3, \ldots, N \) that:

\[ (P) \text{ all new operations } O_{jk} \text{ of operation } O_{\text{com}}[i], k' \text{ can start at the end of } O_{\text{com}}[i], k+1. \]

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 \( J_{\text{com}}[i] \). 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 \( J_{\text{com}}[i] \) move to its next operation, all new operations \( O_{jk} \) can indeed start immediately after \( t \). Four cases are possible: i) only \( O_{[i+1]} \) moves from \( O_{[i+1],k} \) to \( O_{[i+1],k+1} \); ii) only \( J_{\text{com}}[i] \) moves from \( O_{\text{com}}[i,k'] \) to \( O_{\text{com}}[i,k'+1] \); iii) \( J_{[i+1]} \) moves first from \( O_{[i+1],k} \) to \( O_{[i+1],k+1} \) and \( J_{\text{com}}[i] \) moves second from \( O_{\text{com}}[i,k'] \) to \( O_{\text{com}}[i,k'+1] \); and iv) \( J_{\text{com}}[i] \) moves first from \( O_{\text{com}}[i,k'] \) to \( O_{\text{com}}[i,k'+1] \) 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 \( O_{jk} \) related to \( O_{\text{com}}[i,k'+1] \) are not held 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 \( J_{\text{com}}[i] \), resources needed for \( O_{[i+1],k+1} \) are not held by \( O_{\text{com}}[i,k'] \) and hence not hold by operations \( O_{jk} \) related to \( O_{\text{com}}[i,k'+1] \). 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 \( J_{\text{com}}[i] \), resources needed for new operations \( O_{jk} \) related to \( O_{\text{com}}[i,k'+1] \) are not held by \( O_{[i+1],k+1} \) and hence available. By induction assumption, the move from \( O_{\text{com}}[i,k'] \) to \( O_{\text{com}}[i,k'+1] \) 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 meta-heuristics 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 \( \pi_0 \) 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 \( \pi_0 \). 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
on which the jobs are selected gives the initial solution $\pi_0$ of the taboo search. Given a permutation $\pi = J[3]J[2]...J[N]$, 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 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 $N/3$. The attributes of a move of a job $J_i$ from position $x$ to position $y$ recorded in the taboo list are $(i,x)$. 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 multi-resource 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 $M$X 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 I

| model | size | after 5 sec | | | after 60 sec | | | after 600 sec | | |
|-------|------|-------------|-----|-----|-------------|-----|-----|-------------|-----|
|       |      | RND | TS | GA | RND | TS | GA | RND | TS | GA |
| JSPB  | 10 $\times$ 5 | 3 | 1 | 1 | 1 | 2 | 2 | 2 | 5 | 5 |
|       | 10 $\times$ 10 | 0 | 3 | 2 | 1 | 2 | 3 | 3 | 2 | 2 |
|       | 15 $\times$ 10 | 1 | 4 | 0 | 0 | 4 | 1 | 0 | 2 | 3 |
|       | 20 $\times$ 10 | 1 | 2 | 2 | 1 | 2 | 2 | 1 | 2 | 2 |
| Total | 5 | 10 | 5 | 3 | 10 | 8 | 6 | 12 | 12 |
| JMPB  | 10 $\times$ 5 | 2 | 2 | 1 | 1 | 2 | 2 | 3 | 5 | 5 |
|       | 10 $\times$ 10 | 0 | 3 | 2 | 0 | 2 | 3 | 1 | 3 | 2 |
|       | 15 $\times$ 10 | 1 | 3 | 1 | 1 | 2 | 2 | 0 | 3 | 2 |
|       | 20 $\times$ 10 | 1 | 2 | 3 | 0 | 1 | 4 | 1 | 2 | 2 |
| Total | 4 | 9 | 8 | 2 | 7 | 11 | 5 | 13 | 11 |
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. 1.3%). 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 $C_{\text{best}}$ 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 $C_{\text{mean}}$ (resp. $C_{\text{best}}$) 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 $10 \times 5$ instances. The mean relative error between the mean makespan and the best one is about 2.5% for $15 \times 5$ and $20 \times 5$ instances. The mean relative error reaches about 4.3% for $15 \times 10$ 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 $8 \times 5$ (resp. $6 \times 10$) 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>
<thead>
<tr>
<th>Data</th>
<th>$C_{\text{opt}}$</th>
<th>$C_{\text{mean}}$</th>
<th>$C_{\text{best}}$</th>
<th>$C_{\text{mean}}$</th>
<th>$C_{\text{best}}$</th>
<th>$C_{\text{mean}}$</th>
<th>$C_{\text{best}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abz5</td>
<td>1641</td>
<td>1838</td>
<td>1960</td>
<td>1805</td>
<td>1774</td>
<td>1744</td>
<td>1669</td>
</tr>
<tr>
<td>Abz6</td>
<td>1249</td>
<td>1419</td>
<td>1564</td>
<td>1380</td>
<td>1356</td>
<td>1314</td>
<td>1298</td>
</tr>
<tr>
<td>Mt10</td>
<td>1158</td>
<td>1447</td>
<td>1381</td>
<td>1265</td>
<td>1233</td>
<td>1242</td>
<td>1227</td>
</tr>
<tr>
<td>Orb1</td>
<td>1256</td>
<td>1335</td>
<td>1448</td>
<td>1365</td>
<td>1344</td>
<td>1323</td>
<td>1295</td>
</tr>
<tr>
<td>Orb2</td>
<td>1144</td>
<td>1370</td>
<td>1375</td>
<td>1257</td>
<td>1227</td>
<td>1222</td>
<td>1204</td>
</tr>
<tr>
<td>Orb3</td>
<td>1311</td>
<td>$\infty$</td>
<td>1382</td>
<td>1352</td>
<td>1329</td>
<td>1328</td>
<td>1313</td>
</tr>
<tr>
<td>Orb4</td>
<td>1246</td>
<td>1460</td>
<td>1484</td>
<td>1354</td>
<td>1294</td>
<td>1298</td>
<td>1288</td>
</tr>
<tr>
<td>Orb5</td>
<td>1203</td>
<td>1414</td>
<td>1444</td>
<td>1301</td>
<td>1251</td>
<td>1261</td>
<td>1243</td>
</tr>
<tr>
<td>Orb6</td>
<td>1266</td>
<td>1448</td>
<td>1504</td>
<td>1379</td>
<td>1315</td>
<td>1331</td>
<td>1306</td>
</tr>
<tr>
<td>Orb7</td>
<td>527</td>
<td>565</td>
<td>650</td>
<td>574</td>
<td>562</td>
<td>563</td>
<td>556</td>
</tr>
<tr>
<td>Orb8</td>
<td>1139</td>
<td>1223</td>
<td>1223</td>
<td>1183</td>
<td>1166</td>
<td>1160</td>
<td>$*1159$</td>
</tr>
<tr>
<td>Orb9</td>
<td>1130</td>
<td>1239</td>
<td>1422</td>
<td>1248</td>
<td>1226</td>
<td>1175</td>
<td>1162</td>
</tr>
<tr>
<td>Orb10</td>
<td>1367</td>
<td>1789</td>
<td>1711</td>
<td>1485</td>
<td>1447</td>
<td>1468</td>
<td>1447</td>
</tr>
<tr>
<td>La16</td>
<td>1148</td>
<td>1231</td>
<td>1462</td>
<td>1277</td>
<td>1257</td>
<td>1235</td>
<td>1223</td>
</tr>
<tr>
<td>La17</td>
<td>968</td>
<td>1146</td>
<td>1252</td>
<td>1103</td>
<td>1020</td>
<td>1046</td>
<td>1020</td>
</tr>
<tr>
<td>La18</td>
<td>1077</td>
<td>1334</td>
<td>1385</td>
<td>1200</td>
<td>1156</td>
<td>1156</td>
<td>1156</td>
</tr>
<tr>
<td>La19</td>
<td>1102</td>
<td>1314</td>
<td>1392</td>
<td>1249</td>
<td>1228</td>
<td>1221</td>
<td>1210</td>
</tr>
<tr>
<td>La20</td>
<td>1118</td>
<td>1357</td>
<td>1362</td>
<td>1261</td>
<td>1207</td>
<td>1222</td>
<td>1204</td>
</tr>
</tbody>
</table>

| MRE  | 21             | 9.5              | 6.4              | 6               | 4.5            | 4.7             | 3.6              |
Section II with $M = \sum_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}$ 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 $M^X$ 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 $M^X$ gives respectively the best makespan ($C_{\text{max}}$) of our approach and the relative errors $[RE(\%)]$ that are evaluated as follows $100 \times (C_{\text{max}} - Opt)/Opt$ or $100 \times (C_{\text{max}} - LB)/LB$ 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 $M^X$ 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-
tioned before, three sets of test problems with different degree of flexibilities were generated. Each operation is associated on average with machines in a \textit{vdata} instance of high flexibility, with two machines in an \textit{rdata} instance of medium flexibility, and with less than two machines in an \textit{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 are carried out. Table V shows the results in which column , , and 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 \textit{MX}_{\text{mean}} and \textit{MX}_{\text{best}} gives, 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 \textit{MX}_{\text{mean}} and \textit{MX}_{\text{best}} shows that the proposed heuristic \textit{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 \textit{edata}, \textit{rdata} and \textit{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 \textit{edata} instances of [13]. Instances \texttt{La01-La05} contain five resources, while there are ten resources in instances \texttt{La16-La20}. Again, ten independent runs of are carried out with 200 s for each of them. Table VI shows the results, in which column \textit{MX}(0) gives the makespan of the initial solution and column \textit{MX}_{\text{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 \textit{endata} and \textit{erdatal} 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 \textit{evdata} instance, the initial makespan seems good comparing with the mean makespan. For instance, the relative error is less

<table>
<thead>
<tr>
<th>Data</th>
<th>n</th>
<th>m</th>
<th>\textit{edata}</th>
<th>\textit{rdata}</th>
<th>\textit{vdata}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>\textit{MX}(0)</td>
<td>\textit{MX}_{\text{mean}}</td>
<td>\textit{MX}_{\text{best}}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>\textit{MX}(0)</td>
<td>\textit{MX}_{\text{mean}}</td>
<td>\textit{MX}_{\text{best}}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>\textit{MX}(0)</td>
<td>\textit{MX}_{\text{mean}}</td>
<td>\textit{MX}_{\text{best}}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
<th>n</th>
<th>m</th>
<th>\textit{edata}</th>
<th>\textit{erdatal}</th>
<th>\textit{evdata}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>\textit{MX}(0)</td>
<td>\textit{MX}_{\text{mean}}</td>
<td>\textit{MX}_{\text{mean}}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>\textit{MX}(0)</td>
<td>\textit{MX}_{\text{mean}}</td>
<td>\textit{MX}_{\text{mean}}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>\textit{MX}(0)</td>
<td>\textit{MX}_{\text{mean}}</td>
<td>\textit{MX}_{\text{mean}}</td>
</tr>
</tbody>
</table>
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 $v_0 = NW(k_1^0, k_2^0, A_{1,k_1-1}^0, *, *)$, the only modification to introduce to Algorithm A3 concerns the first step. This step becomes $k_1 \leftarrow k_1^0 - 1$, $k_2 \leftarrow k_2^0 + 1$ and the vertical progress is used with $k_1$, $k_2$ and $A_{1,k_2}^0$. 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 $v_0 = SE(k_1^0, k_2^0, *, A_{2,k_2-1}^0)$:

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 $A_{2,k_2-1}$ 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 $AM_{1,k_1}$ or $AM_{2,k_2}$ 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 ($k_1^t \geq n_1$).

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 $v_e = SE(k_1^e, k_2^e, *, A_{2,k_2-1}^e)$. The
case $v_c = NW(k_1, k_2, A_{1,2}^{k_1}A_{2,2}^{k_2}-1)$ is similar. The notation $FEAS(F, A) = \{ A' \in F | A' \cap A = \emptyset \}$ is used.

**Algorithm A4. Resource Assignment Along SP($t_0, v_c$):**

Step 1: Apply Algorithm A3 for $t_0$ by restricting the progress along $SP(t_0, v_c)$. Let $k_1 \leftarrow k_1', k_2 \leftarrow k_2 - 1$.

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

Step 3: Select $A_{1}^{k_1} \in FEAS(A_{1,2}^{k_1}, A_{2,2}^{k_2})$.

Step 4: Backtrack the subpath along the diagonal till hitting a new boundary of a rectangle. If $t_0$ is reached, then stop.

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

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

Step 7: Go to Step 4.

**REFERENCES**


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.