ABSTRACT
The job shop scheduling problem (JSP) arises in low-volume production systems in which products are made to order. In such production systems, jobs usually differ considerably in their processing sequences and times. Solving the JSP is particularly important for the efficient utilization of production resources and for satisfying due dates. However, the JSP is fairly complex and known to be NP-hard. This complexity demands continuous efforts for developing efficient solution approaches. This paper suggests a new network model for the JSP which can be used in future exact and approximate algorithmic developments. The new network model is experimentally compared with the well known disjunctive graph model through an exact branch-and-bound algorithmic implementation. The experimental results indicate that there are some merits in the new model that may guide a search algorithm to quickly reach an optimal solution.

KEYWORDS
Job shop scheduling, disjunctive graph model, branch-and-bound, exact algorithms

1. INTRODUCTION
1.1. Problem definition and notations
The JSP is concerned with sequencing a set of jobs, $J$, on a set $M$ of technologically different machines; each is capable of processing at most one job at a time. Jobs follow dissimilar processing routes among the machines and a job can not be processed on more than one machine simultaneously. Furthermore, preemption is not allowed and a job is permitted to have multiple visits to any machine. This paper addresses the static, deterministic version of the problem in which raw materials for all jobs are assumed to be ready for processing at the beginning of the schedule and the processing times are deterministic.

In the JSP, each job consists of an ordered list of operations that represents its processing route. We denote $I = \{1, 2, \ldots, \nu\}$ as the set of all operations’ indexes. The operations’ indexes are assigned such that for job $k \in J$, the subset of consecutive indexes $I_k = \{\alpha_k, \alpha_k + 1, \alpha_k + 2, \ldots, \alpha_k\} \subseteq I$ includes the indexes of operations that belong to that job; where in the set $I_k$, the operation with the lower index is to be processed first. For operation $i$, the time needed to finish its processing is $p_i$, the job to which it belongs is denoted $jb(i)$, and its machine is denoted $mc(i)$.

The task of the scheduling process is to determine the start time $s_i$ for every operation $i \in I$. There are two sets of constraints in the JSP. The technological or precedence constraints define the mandatory processing sequence of operations belonging to the same job. This set of constraints is represented by the following inequalities.

\begin{equation}
    s_{i+1} - s_i \geq p_i \quad \forall \; i, i+1 \in I_k \quad \forall \; k \in J
\end{equation}

The second set of constraints is in a disjunctive (either-or) form, and it represents the condition that processing of different operations on the same machine must be conducted in different time intervals.

\begin{equation}
    s_i \geq s_j + p_j \quad \text{or} \quad s_j \geq s_i + p_i \quad \forall \; i, j \in I, \text{ where } mc(i) = mc(j) \text{ and } jb(i) \neq jb(j)
\end{equation}

Different objective functions have been dealt with in the literature of the JSP. In this paper, we concentrate on the objective of minimizing the maximum completion time or the makespan. The makespan, denoted $C_{\text{max}}$, is expressed as follows.

\begin{equation}
    C_{\text{max}} = \max_{i \in I, j \in J} \{s_i + p_j\}
\end{equation}

Even for the small case of three jobs and three machines, the JSP, with the objective of minimizing the makespan, is known to be NP-hard (Sotskov and Shakhlevich, 1995). Earlier complexity results for the JSP are provided by Garey et al (1976). This high
level of complexity demands continuous efforts for developing efficient solution approaches and clever mathematical representations that can provide effective guidance to solution approaches.

1.2. The disjunctive graph model

The disjunctive graph model, developed by Roy and Sussman (1964), has been used as a standard network representation for the job shop scheduling problem. Based on the same concepts found in the structure of the activity-on-arc project networks, nodes in the disjunctive graph model are used to represent the event of starting the processing of an operation. Here, we use operation indexes as labels for nodes. An arc connecting two consecutive nodes \( i \) and \( i+1 \), where operations \( i \) and \( i+1 \in I_k \) for some job \( k \), represents the activity of processing operation \( i \), and has a length of \( p_i \). Using the language of project scheduling, the operations belonging to the same job are represented as a series of consecutive activities. In addition, two dummy, source \((0)\) and terminal \((\nu+1)\), nodes are defined to respectively represent the events of starting and ending the schedule. To complete the project network, dummy arcs, \((0, \alpha_i)\) and \((\omega_k, \nu+1)\) for all \( k \in J \), with zero lengths are added. The project network \( N = (V; Z) \), for the set of nodes \( V = \{i; i = 0, ..., \nu+1\} \) and the set of directed arcs \( Z = \bigcup_{k \in J} Z_k \), where \( Z_k = \{(i, i+1); i, i+1 \in I_k\} \cup \{(0, \alpha_k), (\omega_k, \nu+1)\} \), is sufficient to represent the technological constraints (1); however, it is not a complete representation for the JSP as constraints (2) are not taken into consideration.

To represent constraints (2), disjunctive pairs of arcs are defined over the nodes of all the operations that share the same machine and belong to different jobs. The disjunctive pair of arcs between nodes \( i \) and \( j \) on a graph, written as \( \langle i, j \rangle \), is formed by the two arcs \((\omega_i, \alpha_j)\) and \((\alpha_i, \omega_j)\). The selection of the disjunctive pair \( \langle i, j \rangle \), denoted \( Sel((i, j)) \), is an arc, either \((i, j)\) with length \( p_i \), or \((j, i)\) with length \( p_j \). The selection \((i, j)\) is called the complement of the selection \((j, i)\) and visa versa. A predetermined selection \( Sel((i, j)) = (i, j) \) is said to be complemented when it is changed to \((j, i)\). A complete selection in a disjunctive graph is obtained when selections for all disjunctive pairs of arcs are decided. The disjunctive graph is defined as \( G = (V; Z, W) \) for the set of disjunctive pairs \( W = \bigcup_{m \in \mathbb{N}} W_m \), where \( W_m = \{(i, j); i, j \in Q_m \text{ and } jb(i) \neq jb(j)\} \) for the set of nodes \( Q_m = \{i; i \in I \text{ and } mc(i) = m\} \).

In the disjunctive graph model, each disjunctive arc is associated with a binary decision variable whose value determines its selection. Solution algorithms that are built upon that model operate in the domains of these binary decision variables to determine a complete selection on the disjunctive graph. A complete selection should not result in a cyclic graph so as to be able to interpret it into a unique feasible schedule.

One of the early exact solution algorithms that are based on the disjunctive graph model is the implicit enumeration algorithm of Balas (1969). However, the majority of the disjunctive graph-based solution algorithms are branch-and-bound (BAB). Examples in the literature include: Charlton and Death (1970), Carlier and Pinson (1989), Applegate and Cook (1991) and Brucker et al (1994). Jain and Meera (1999) provide a recent review. It is known that the most effective BAB methods for the JSP are based on the disjunctive graph model (Brucker, 2001). To the best of our knowledge, the exact solution algorithms that are based on the disjunctive graph model consider solely the objective of minimizing the makespan. The reason for that may be due to the similarity between the network structure in the disjunctive graph model and project networks. This similarity facilitates the use of the critical path algorithm, which results in determining the critical path length or equivalently the makespan when a complete selection is obtained. Furthermore, in the case of incomplete selection, the critical path algorithm is useful in evaluating a lower bound associated with the determined selections.

An apparent shortcoming of the disjunctive graph model is that the worst-case number of disjunctive selection decisions is growing with a rate greater than the number of operations’ permutations, which is sufficient to represent all feasible schedules. To illustrate, consider the case of sequencing \( b \) distinct operations belonging to different jobs on a single machine as part of a larger JSP instance. The number of disjunctive pairs of arcs is given by \( \sum_{i=1}^{b-1} (b-i) = b(b-1)/2 \). Since there are two distinct selections for each pair of disjunctive arcs, the worst-case number of selections that need to be investigated is \( 2^{b(b-1)/2} \). On the other hand, all feasible schedules can be simply generated by enumerating all job permutations. The number of job permutations for the single machine, \( b \) operations case is \( b! \). Using
mathematical induction, it can be shown that \( b! < 2^{(b-1)^2} \) for all \( b > 2 \). In fact, the quantity \( (2^{(b-1)^2} - b!) \) is the number of complete selections that result in directed graphs that contain cycles formed by arcs connecting any subset of three or more nodes.

Even though the existing exact algorithms that are based on the disjunctive graph model use smart rules to avoid selections that result in cycles, the huge size of the search space of the binary decision variables remains questionable. In this paper, we present an alternate network model for the JSP that is based on job permutations as decision variables. These job permutations are translated into directed arcs which are added to the network \( N \). We refer to this network model as the Permutation-Induced Acyclic Network (PIAN). We compare the two models through an algorithmic implementation that is based on branch-and-bound.

1.3. Outline of the paper

The rest of this paper is organized as follows. In section 2, the new network model along with its theoretical properties is presented. In sections 3, we present a simple branch-and-bound (BAB) algorithmic implementation that can be applied to both models. In section 4, we present and discuss the experimental results. Finally, the conclusion is provided in section 5.

2. PERMUTATION-INDUCED ACYCLIC NETWORK (PIAN)

2.1. Basic definitions and properties

We define a processing sequence for a given machine \( m \), denoted \( \pi_m \), as a permutation representing the order by which operations are to be processed. A processing sequence is formed by a subset of the operations that need to be processed on a given machine, and it is said to be complete if all the operations are included. We denote \( i \in \pi \) if operation \( i \) is included in the processing sequence \( \pi \), and \( i \notin \pi \) if not. The number of operations included in \( \pi_m \) is denoted \( |\pi_m| \). The vector of processing sequences on all machines is denoted \( \Pi = (\pi_m : m \in M) \). In the new network model, the vector of complete processing sequences, \( \Pi^c = (\pi_m^c : m \in M) \), is the decision variable that needs to be determined.

Let \( [i]^{(\pi_m)} \) be an integer that takes a value from 1 to \( |\pi_m| \) and represents the processing order of operation \( i \) in \( \pi_m \), provided that \( mc(i) = m \) and \( i \in \pi_m \). Next, we present two definitions that characterize a relationship property between different processing sequences.

Definition 1. For a given machine \( m \), a processing sequence \( \pi_m \) is said to be component of another processing sequence \( \pi_m' \), written as \( \pi_m' < \pi_m \), if and only if for every operation \( i \in \pi_m \), it should be \( i \in \pi_m' \) and for any two distinct operations \( i \) and \( j \in \pi_m \), if \( [i]^{(\pi_m)} < [j]^{(\pi_m)} \), it should be \( [i]^{(\pi_m')} < [j]^{(\pi_m')} \).

Definition 2. A vector of processing sequences \( \Pi^c = (\pi_m^c : m \in M) \) is said to be component of another vector \( \Pi = (\pi_m : m \in M) \), written as \( \Pi^c < \Pi \), if \( \pi_m^c < \pi_m \) for all \( m \in M \).

The set of vectors of processing sequences that are derived from a given vector of processing sequences \( \Pi \) is denoted \( \Delta(\Pi) = [\Pi^c : \Pi < \Pi^c]. \) Consequently, for a vector of complete processing sequences \( \Pi^c \), \( \Delta(\Pi^c) = \phi. \)

As in the disjunctive graph model, the PIAN model uses operations’ indexes as labels for nodes, where the node labeled \( i \) represents the event of starting the processing of operation \( i \). Similarly, two dummy nodes, 0 and \( \nu + 1 \), are defined, and the set of arcs \( Z \) is used to represent the technological constraints. Instead of using disjunctive arcs, we define a processing sequence arc as follows.

Definition 3. For a given processing sequence \( \pi_m \), a processing sequence arc between two distinct nodes \( i \) and \( j \), written as \( (i, j)^{\pi_m} \), is defined as:

\[
(i, j)^{\pi_m} = \begin{cases} 
\text{arc}(i, j) \text{ with length } p_{ij}, & \text{if } [i]^{(\pi_m)} < [j]^{(\pi_m)} \\
\text{arc}(j, i) \text{ with length } p_{ij}, & \text{if } [i]^{(\pi_m)} > [j]^{(\pi_m)} \\
\phi & \text{if either } i \notin \pi_m \text{ or } j \notin \pi_m
\end{cases}
\]

We construct the set of arcs \( A_m(\pi_m) = \{(i, j)^{\pi_m} : i, j \in Q_m \text{ and } jb(i) \neq jb(j) \} \) for the set of nodes \( Q_m = \{ i: i \in I \text{ and } mc(i) = m \} \). The resultant digraph is denoted \( \Gamma_m(\pi_m) = (Q_m, A_m(\pi_m)) \). The following lemma states an important characteristic for this digraph.
Lemma 1. For any given processing sequence \( \pi_m \), the digraph \( \Gamma_m(\pi_m) \) is acyclic.

For a given machine \( m \), there is a one-to-one correspondence between the complete selections that do not result in cycles in the disjunctive graph model and the digraph \( \Gamma_m(\pi_m) \) when the processing sequence \( \pi_m \) is complete. The following lemma emphasizes this relationship.

**Lemma 2.** In the disjunctive graph model, if the set of arcs of a complete selection, \( S_m = \{ \text{Sel}(x) : x \in W_m \} \), forms an acyclic digraph \( g_m = (Q_m; S_m) \), then \( g_m = \Gamma_m(\pi_m) \) for one and only one complete processing sequence \( \pi_m \).

For a given vector of complete processing sequences \( \Pi = (\pi_m : m \in M) \), the resultant network is defined as \( \Gamma(\Pi) = (V; Z \cup \bigcup_{m \in M} A_m(\pi_m)) \). In order for this network to represent a feasible schedule, the condition given in the following proposition should be satisfied. There is a similarity between this condition and a known, unproven fact for the disjunctive graph model which states that complete selections should not result in cycles in order to obtain feasible schedules (Pinedo, 2002).

**Proposition 1.** A vector of complete processing sequences on all the machines, \( \Pi \), constitute feasible schedule if the resultant digraph \( \Gamma(\Pi) \) is acyclic.

**Proof.** As indicated in lemma 1, for a given machine \( m \), a complete processing sequence, \( \pi_m \), results in an acyclic digraph \( \Gamma_m(\pi_m) \). So, we start with the case in which the digraph \( \Gamma(\Pi) = (V; Z \cup \bigcup_{m \in M} A_m(\pi_m)) \) is acyclic. The following two cases define the conditions for which there is a conflict between the operations’ processing sequences on machines and the technological constraints, which result in infeasible solutions.

Case 1: \( i, j \in I_m, mc(i) = mc(j) = m, i < j \) and for the processing sequence \( \pi_m, [j]^{\pi_m} < [i]^{\pi_m} \).

Case 2: \( i_1, j_1 \in I_{m_1}, i_2, j_2 \in I_{m_2}, k_1 \neq k_2, mc(i_1) = mc(j_2) = m_1, mc(i_2) = mc(j_1) = m_2, m_1 \neq m_2, i_1 < j_1, i_2 < j_2 \), and for the given processing sequence \( \pi_{m_1}, [j_2]^{\pi_{m_1}} < [i_1]^{\pi_{m_1}} \) and for \( \pi_{m_2}, [j_1]^{\pi_{m_2}} < [i_2]^{\pi_{m_2}} \).

It can be easily shown that in both cases, the digraph \( \Gamma(\Pi) \) that result from adding the set of arcs \( Z \) to \( \Gamma(\Pi) \) will contain cycles. Therefore, the digraph \( \Gamma(\Pi) \) should be acyclic in order to obtain feasible schedules on machines.

Since the exact solution algorithms that are based on the disjunctive graph model consider solely the objective of minimizing the makespan, the rest of this paper will focus on that objective. For a given vector of processing sequences, \( \Pi = (\pi_m : m \in M) \), let \( C(\Pi) \) be the longest path length from node 0 to node \( \nu + 1 \) on the network \( \Gamma(\Pi) \). As it is commonly known in the literature of the disjunctive graph model for the case of complete selections (Pinedo, 2002), the value of \( C(\Pi) \) is the makespan of the schedule defined by \( \Pi \) whenever its coordinates are complete, given that the condition of proposition 1 is satisfied.

### 2.2. Elements of the PIAN model

The elements of the PIAN model as a representation to the JSP are described as follows. Similar to the disjunctive graph model, the structure of jobs along with the technological or precedence constraints are represented by the network \( N \). The set of complete processing sequences on all machines defines the search space in the PIAN model. The operations’ requirements of machines and the disjunctive constraints need not to be explicitly stated as they are inherent in the definition of the domains of the complete processing sequence decision variables. The only constraint on the complete processing sequences is that they have to be selected such that proposition 1 is satisfied. The calculation of the makespan objective can be done by evaluating the critical path length \( C(\Pi) \) for any given vector of complete processing sequences \( \Pi = (\pi_m : m \in M) \). The determination of the operations’ start times can be done easily by a simple algorithm that interprets complete processing sequences on machines into a semi-active schedule.

### 2.3. Generating lower bounds

Based on the disjunctive graph model, the lower bound generating procedures for the JSP employ a longest path algorithm. These procedures can still be applied to the PIAN model. The simplest lower bound for a given vector of processing sequences \( \Pi \), is \( C(\Pi) \). More sophisticated lower bounds are based on evaluating head and tail values for each node on the graph. For node \( i \), a head, denoted \( r_i \), is used to represent the earliest start time of operation \( i \), and a
tail, denoted \( q_n \), is used to represent the minimum length of time needed for the schedule to finish after the processing of this operation finishes. If \( L^{(i)}(i,j) \) denotes the length of the longest path from node \( i \) to node \( j \) on the network \( \Gamma(\Pi) \), the head and tail values can be evaluated respectively as \( r_i = L^{(i)}(0,i) \) and \( q_j = L^{(j)}(i,v) - p_j \). A more sophisticated method for calculating heads and tails is described in Brucker (2001). Using the calculated heads and tails, the lower bound is evaluated by solving a single-machine Jackson's preemptive schedule for each machine (Carlier, 1982). The maximal makespan of these schedules is a lower bound of the JSP. We denote \( LB(\Pi) \) as the lower bound generated using this procedure for the vector of processing sequences \( \Pi \). The following theorem states an important property for such a lower bound.

**Theorem 1.** For any two vectors of processing sequences \( \Pi_1 \) and \( \Pi_2 \), if \( \Pi_1 \preceq \Pi_2 \), then \( LB(\Pi_1) \leq LB(\Pi_2) \).

**Proof.** Since \( \Pi_1 \preceq \Pi_2 \), the network \( \Gamma(\Pi_2) \) contains exactly the same arcs, with the same lengths, as in the network \( \Gamma(\Pi_1) \) in addition to other arcs that result from the additional operations included in \( \Pi_2 \) and not included in \( \Pi_1 \). Therefore, for any nodes \( i \) and \( j \), it should be \( L^{(i)}(i,j) \leq L^{(j)}(i,j) \). Accordingly, the head of any operation in \( \Pi_1 \) will be at least the same as that in \( \Pi_2 \). It can be easily shown that the lower bound generated using Jackson's preemptive schedule is monotone increasing function of the operations' heads. Therefore, it should be \( LB(\Pi_1) \leq LB(\Pi_2) \).

**Corollary 1.** For a given vector of processing sequences \( \Pi \) in which there is at least one incomplete processing sequence, \( LB(\Pi) \leq LB(\Pi') \) for all \( \Pi' \in \Delta(\Pi) \).

As a result of corollary 1, solutions can be constructed in a BAB framework by inserting new operations to the existing processing sequence of a given node to generate child nodes.

### 3. SIMPLE BAB ALGORITHMIC IMPLEMENTATION

The apparent advantage of the PIAN model over the disjunctive graph model is that the size of the search space for the binary decision variables of the later is greater than that of the complete processing sequence decision variables of the former. The size of the search space may provide a measure for the worst-case complexity of an exact algorithm; however, it is not sufficient to justify the superiority of one model over the other. The efficiency of the solution algorithms that operate in the domains of the decision variables of a given model is what matters. This efficiency may be measured by the capability of a solution algorithm to quickly determine the right directions in the search space that lead to an optimal solution and to avoid those directions that lead to inefficient or suboptimal solutions and consume more time.

In order to assess the significance of the new model and to examine the algorithmic benefits from its representation to the JSP, we use a simple depth-first BAB implementation that has similar search mechanism for both models. The similarity in the search mechanism is achieved by using the same method for evaluating lower and upper bounds at each node of the search tree, and by using similar rules for selecting the branching operation or disjunctive arc. The following two lists illustrate the main steps of this BAB algorithm. The disjunctive graph-based implementation uses branching over disjunctive selections of a candidate disjunctive arc, while the second uses branching over the positions in which a candidate operation is to be placed in a processing sequence. However, both implementations follow quite similar rules for selecting the candidate disjunctive arc and the candidate operation on which branching is conducted.

**Disjunctive graph-based simple branch-and-bound algorithm**

1. **Evaluate a starting upper bound:** Construct a feasible active schedule using Giffler and Thompson (1960) algorithm with the most work remaining (MWR) dispatching rule. Let \( S^* \) denote this schedule, and \( UB \) equal to its makespan.
2. **Initialize:** Set the branch-and-bound tree node index \( u = 0 \), and the branch-and-bound tree node counter \( nc = 1 \). Start with an empty set of selections \( SL_0 = \emptyset \).
3. **Calculate bounds at the current node \( u \):**
   3.1. Construct the network \( NW_u = (V; Z \cup SL_u) \).
3.2. Use a longest path algorithm to calculate the heads \( r_i \) and tails \( q_i \) for all operations based on the network \( NW_u \).

3.3. Let the lower bound at node \( u \), \( LB_u \), equal to the lower bound generated by Jackson's preemptive schedule using the calculated heads and tails.

3.4. If \( LB_u \geq UB \), proceed to step 5.

3.5. Construct two semi-active schedules: \( S^1 \) using a non-decreasing order of the operations' heads, and \( S^2 \) using a non-increasing order of the operations' tails. From \( S^1 \) and \( S^2 \), let \( S_o \) denote the selected schedule having the minimum makespan and \( UB_o \), its makespan.

3.6. If \( UB_o < UB \), let \( S^* = S_o \) and \( UB = UB_o \).

3.7. If \( UB = LB_o \), STOP an optimal solution, \( S^* \), is found.

4. Prepare for branching and generate first child node:

4.1. Let \( s_i = r_i \) for all \( i \in I \). Select a disjunctive arc \( <x, y> \) for which the disjunctive constraint is not satisfied such that \( s_x \) and \( s_y \) are the minimum and in case of tie-breaking, priority is given to the operation with the smallest processing time. If no such a disjunctive arc is found, proceed to step 5.

4.2. Let \( sa_u \) denote a selection made at node \( u \), \( Sel(<x, y>) \), for branching. If \( s_x \leq s_y \), let \( sa_u = (x, y) \); otherwise, let \( sa_u = (y, x) \).

4.3. Let \( u = u + 1 \) and \( nc = nc + 1 \).

4.4. Let \( SL_u = SL_{u-1} \cup sa_u \).

4.5. Proceed to step 3.

5. Backtrack and generate uninvestigated nodes:

5.1. Let \( u \) be the index of the nearest parent branch-and-bound tree node whose selection has not been complemented. If no such a node can be found, STOP. \( S^* \) is an optimal solution.

5.2. Let \( \overline{sa_u} \) be the complement of \( sa_u \). Label the selection at node \( u \) as complemented.

5.3. Let \( SL_u = SL_u \cup \overline{sa_u} \).

5.4. Let \( u = nc \), and \( nc = nc + 1 \). Proceed to step 3.

PIAN-based simple branch-and-bound algorithm

1. Evaluate a starting upper bound: Construct a feasible active schedule using Giffler and Thompson (1960) algorithm with the most work remaining (MWR) dispatching rule. Let \( S^* \) denote this schedule, and \( UB \) equal to its makespan.

2. Initialize: Set the branch-and-bound tree node index \( u = 0 \), and the branch-and-bound tree node counter \( nc = 1 \). Construct the initial vector of processing sequences \( \Pi_0 = (\pi_{0,m} : m \in M) \); where each \( \pi_{0,m} \) contains exactly one operation, \( o_m \), selected from the sub-set \( I_m \subset I \) which is defined as the subset of operations that has the minimum value of \( i - \alpha_s \), where \( mc(i) = m \) and \( k = jb(i) \). This operation is arbitrarily selected as \( o_m = \min \{ i : i \in I_m \} \).

3. Calculate bounds at the current node \( u \):

3.1. Construct the network \( NW_u = \Gamma(\Pi_u) \).

3.2. Use a longest path algorithm to calculate the heads \( r_i \) and tails \( q_i \) for all operations based on the network \( NW_u \). During the calculation of heads and tails, if \( NW_u \) is found to contain cycles, proceed to step 5.

3.3. Let the lower bound at node \( u \), \( LB_u \), equal to the lower bound generated by Jackson's preemptive schedule using the calculated heads and tails.

3.4. If \( LB_u \geq UB \), proceed to step 5.

3.5. Construct two semi-active schedules: \( S^1 \) using a non-decreasing order of the operations' heads, and \( S^2 \) using a non-increasing order of the operations' tails. From \( S^1 \) and \( S^2 \), let \( S_o \) denote the selected schedule having the minimum makespan and \( UB_o \), its makespan.

3.6. If \( UB_o < UB \), let \( S^* = S_o \) and \( UB = UB_o \).

3.7. If \( UB = LB_o \), STOP an optimal solution, \( S^* \), is found.

4. Prepare for branching and generate first child node:

4.1. Let \( s_i = r_i \) for all \( i \in I \). Select a branching operation \( b_o \notin \pi_{u,m} \) for all \( m \in M \), such that there exists an operation \( i \), where \( mc(i) = mc(b_o) \), for which the disjunctive constraint with \( b_o \) is not satisfied, and \( s_{b_o} \) is the minimum. In case of tie-breaking, priority is given to the operation with the smallest processing time. If no such an operation is found, proceed to step 5.

4.2. Select a starting position \( sp_o \) for placing operation \( b_o \) in \( \pi_{u,m} \) where \( sp_o \) is determined by investigating the set \( Pre_{s_o} = \{ [i]^{\pi_{u,m}} + 1 : i \in \pi_{u,m} \) and \( s_i < s_{b_o} \} \). If

\[ s_i < s_{b_o} \]
The disjunctive graph-based BAB implementation is based on the one proposed by Charlton and Death (1970). Using the calculated heads as the start times for operations, the candidate disjunctive arc on which branching is conducted is selected from among those whose corresponding disjunctive constraints (2) are not satisfied. Preference in selection is made to those disjunctive arcs that are connecting operations that have the minimum heads and shortest processing times. It is shown in Bellman et al (1982) that this selection strategy guarantees that the resultant graph is acyclic. Accordingly, there is no need to check for cycles.

In the PIAN-based BAB implementation, the candidate operation selected for branching is chosen in a similar fashion as done in the disjunctive graph-based version of the algorithm. Heads are used as values for the start times of the operations. Based on that, the algorithm looks for an operation with the minimum head and has at least one unsatisfied disjunctive constraint with any of the operations that need to be processed on the same machine. Preference is made to the operation with the shortest processing time. Then the starting position in which the selected operation is to be placed is chosen such that the operations in the corresponding processing sequence are sorted in a non-decreasing order of their heads or start times. As shown in steps 5.2 to 5.4, all the remaining positions are then investigated using forward and then backward moves from the starting position. The use of this position selection strategy is found to reduce the number of investigated BAB nodes as better upper bound can be obtained early at the starting position.

In the following section, we present the experimental results obtained for 80 randomly generated test problems. In comparing the two implementations, we are concerned with the total number of BAB tree nodes investigated until a proven optimal solution is reached. That is the BAB tree node counter nc. This number is in fact a good representative for the computational time of the algorithm since the main computational burden in both implementations is in calculating heads and tails and evaluating the lower and upper bounds at each node of the search tree, which are the same in both implementations.

4. EXPERIMENTAL RESULTS AND DISCUSSION

The two BAB algorithmic implementations are programmed using Borland C++ Builder version 5 and run under Windows XP professional. A set of 80 randomly generated test instances have been used to compare both implementations. These test instances are generated using the same approach provided by Taillard (1993) but with different number of jobs and number of machines.

We opted to use small sized test instances in order to be able to obtain proven optimal solutions in small
computational time. Yet, the comparison between both implementations is not based on the computational time expressed in terms of the actual CPU clock time. Rather, the total number of the investigated BAB tree nodes until a guaranteed optimal solution is obtained (\( n_c \)) is reported. This figure is referred to as \( n_c^{\text{Disj}} \) for the disjunctive graph-based implementation and \( n_c^{\text{PIAN}} \) for the PIAN-based implementation.

Out of 80 test instances, in 26 cases, the PIAN-based implementation was capable of generating optimal solutions (or 3% optimality gap for only two cases) in less number of BAB tree nodes versus 39 cases for the disjunctive graph based implementation. The logarithmic value to the base 10 of the ratio \( n_c^{\text{PIAN}} / n_c^{\text{Disj}} \) is calculated to provide a relative measure for the number of nodes investigated in both implementations.

Positive values for \( \log(n_c^{\text{PIAN}} / n_c^{\text{Disj}}) \) correspond to problem instances in which the performance of the disjunctive graph-based algorithm is better than the PIAN-based. Conversely, negative values for \( \log(n_c^{\text{PIAN}} / n_c^{\text{Disj}}) \) indicate better performance for the PIAN-based algorithm. The benefit of using the log function is that the magnitude of its value can provide a consistent measure of how good an algorithm is relative to the other.

As illustrated in Figure 1, The distribution of the value of \( \log(n_c^{\text{PIAN}} / n_c^{\text{Disj}}) \) suggests that the disjunctive graph-based implementation has on average better performance compared to the PIAN-based implementation. Nevertheless, the existence of instances for which \( \log(n_c^{\text{PIAN}} / n_c^{\text{Disj}}) \) has a relatively high magnitude (>1.0) suggests that there are some characteristics in each model that make it a lot faster than the other in reaching better upper bounds and accordingly a proven optimal solution. Accordingly, one can not claim the superiority of one model over the other under the used simple BAB algorithm.

5. CONCLUSION

The purpose of this paper is to present a new network model for the job shop scheduling problem. The new network model has a structure similar to the traditional disjunctive graph model, except that job permutations on machines are used as decision variables instead of the disjunctive binary decision variables. In the new model, job permutations are interpreted into directed arcs in the network allowing for the use of the critical path length algorithm to evaluate the makespan of the schedule represented by given job permutations, and consequently evaluating lower bounds in a similar way as in the disjunctive graph model.

The apparent benefit of using the new network model over the traditional disjunctive graph model is that the size of the search space for the decision variables is significantly reduced. This reduction in the size of the search space suggests that the worst-case performance of an exact solution algorithm that is
based on the new model will be better than that of the disjunctive graph model. However, this is not enough to claim the superiority of the new model. In order to assess the significance of the new model from the side of the algorithmic implementation, the performance of simple exact BAB algorithm that is based on both network models is compared.

The experimental results show that, on average, the performance of the BAB implementation that is based on the disjunctive graph model is better than that of the PIAN-based implementation. Nevertheless, there are some merits in the new model that allow it to achieve a proven optimal solution in much less computational time.

As part of future research, additional properties of the PIAN model need to be investigated. Such properties might help in speeding up the solution algorithms and might allow for implementing more sophisticated techniques similar to those found in the literature of the disjunctive graph model such as immediate selections.

REFERENCES


