Searching for Optimal Schedules to the Job-Shop Problem with Operators

Paper 53

Abstract

We confront the job-shop scheduling problem with operators. This is a recently proposed problem motivated by manufacturing processes where each operation has to be assisted by one human operator and the number of available operators is limited. We propose a scheduling generation scheme for this problem that extends the well-known $G\&T$ algorithm. This scheme is then exploited to design an any-time algorithm that combines best-first and greedy search. In order to guide the search, two monotonic heuristics are devised from problem relaxations. Also, a method for pruning states based on dominance relations is proposed. The algorithm is evaluated across several benchmarks and compared with other approaches. The results of the experimental study show that our approach clearly outperforms the state-of-the-art methods.

Introduction

We propose a new exact approach to the job-shop scheduling problem with operators which is based on best-first heuristic search. This problem has been recently proposed in (Agnetis et al. 2010) and it is motivated by manufacturing processes in which part of the work is done by human operators sharing the same set of tools. The problem is formalized as a classical job-shop problem in which the processing of an operation on a given machine requires the assistance of one of the p available operators. In (Agnetis et al. 2010) the problem is studied and the minimal NP-hard cases are established. Also, a number of exact and approximate algorithms to cope with this problem are proposed and evaluated on a set of instances generated from that minimal relevant cases. The result of their experimental study makes it clear that instances with 3 jobs, 3 machines, 2 operators and a number of 30 operations per job are really hard to solve to optimality.

The main contribution of the present paper is the definition and study of a new schedule generation scheme that is inspired in the $G\&T$ algorithm proposed in (Giffler and Thompson 1960) for the classical job-shop scheduling problem. This new scheme is then exploited to devise a best-first search algorithm and also a greedy algorithm. These algorithms are used in combination and their performance relies on two heuristic estimations and a rule for pruning search states that are also proposed in this paper. We have conducted an experimental study across instances of different sizes and characteristics. The results of this study show that our approach outperforms the state-of-the-art algorithms, that as far as we know are those proposed in (Agnetis et al. 2010).

The remaining of the paper is organized as follows. In the next section we define the problem and propose a disjunctive model for it. Then we describe and study the new schedule generation scheme termed $OG\&T$. After that, the new best-first search and greedy algorithms are described. This section includes the heuristics defined from two different problem relaxations and the pruning method based on dominance relations among states. The subsequent section is devoted to the experimental study and finally we summarize the main conclusions of the paper and propose some ideas for future research.

Problem Formulation

Formally the job-shop scheduling problem with operators can be defined as follows. We are given a set of n jobs $\{J_1, \ldots, J_n\}$, a set of m resources or machines $\{R_1, \ldots, R_m\}$ and a set of p operators $\{O_1, \ldots, O_p\}$. Each job J_i consists of a sequence of v_i operations or tasks $(\theta_{i1}, \ldots, \theta_{iv_i})$. Each task θ_{il} has a single resource requirement $R_{\theta_{il}}$, an integer duration $p_{\theta_{il}}$ and a start time $st_{\theta_{il}}$ to be determined. A feasible schedule is a complete assignment of starting times and operators to operations that satisfies the following constraints: (i) the operations of each job are sequentially scheduled, (ii) each machine can process at most one operation at any time, (iii) no preemption is allowed and (iv) each operation is assisted by one operator and one operator cannot assist more than one operation at a time. The objective is finding a feasible schedule that minimizes the completion time of all the operations, i.e. the makespan. This problem was first defined in (Agnetis et al. 2010) and is denoted as $JSO(n, p)$.

The significant cases of this problem are those with $p < min(n, m)$, otherwise the problem is a standard job-shop problem denoted as $J||C_{max}$.

Scheduling problems are usually represented by means of a disjunctive model. We propose here to use the following model for the $JSO(n, p)$ that is similar to that used in (Agnetis et al. 2010). A problem instance is represented by a

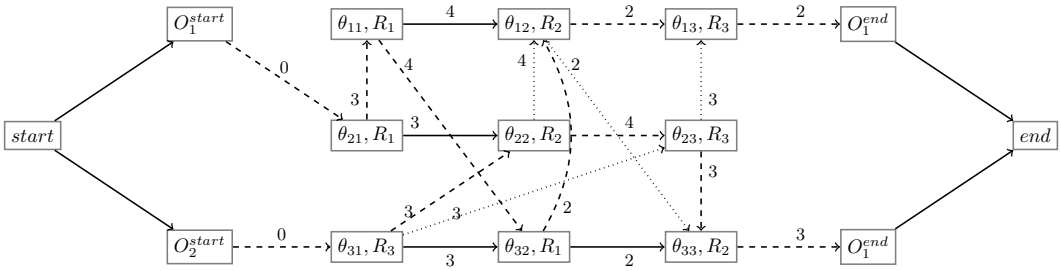

Figure 1: A feasible schedule to a problem with 3 jobs, 3 machines and 2 operators.

directed graph $G = (V, A \cup E \cup I \cup O)$. Each node in the set V represents either an actual operation, or any of the fictitious operations with null processing time introduced with the purpose of giving the graph a particular structure. These fictitious operations include starting and finishing operations for each operator i, denoted O_i^{start} and O_i^{end} respectively, and the the dummy operations *start* and *end*.

The arcs in A are called *conjunctive arcs* and represent precedence constraints among operations of the same job. The arcs in E are called *disjunctive arcs* and represent capacity constraints. E is partitioned into subsets E_i with $E = \cup_{\{i=1,\ldots,M\}} E_i$. E_i includes an arc (v, w) for each pair of operations requiring the resource R_i. The set O of *operator arcs* includes three types of arcs: one arc (u, v) for each pair of operations of the problem, and arcs (O_i^{start}, u) and (u, O_i^{end}) for each operator node and operation. The set I includes arcs connecting node *start* to each node O_i^{start} and arcs connecting each node O_i^{end} to node *end*. The arcs are weighted with the processing time of the operation at the source node.

From this representation, building a solution can be viewed as a process of fixing disjunctive and operator arcs. A disjunctive arc between operations u and v gets fixed when one of (u, v) or (v, u) is selected and consequently the other one is discarded. An operator arc between u and v is fixed when (u, v), (v, u) or none of them is selected, and fixing the arc (O_i^{start}, u) means discarding (O_i^{start}, v) for any operation v other than u. Analogously for (u, O_i^{end}).

Therefore, a feasible schedule S is represented by an acyclic subgraph of G, of the form $G_S = (V, A \cup H \cup I \cup Q)$, where H expresses the processing order of operations on the machines and Q expresses the sequences of operations that are assisted by each operator. In other words, $H = \cup_{i=1,\ldots,m} H_i$, H_i being a subset of E_i that includes the arc (u, v) iff u is processed before v in S. $Q = \cup_{i=1,\ldots,p} Q_i$, where Q_i is a subset of Q that represents the order in which the k_i operations assisted by the operator i are processed. So, it includes the arcs (O_i^{start}, u_1^i), $(u_{k_i}^i, O_i^{end})$ and (u, v) for each pair of operations assisted by the operator i such that u is processed before v. For each operation u, at least two arcs of the form $(u, -)$ and $(-, u)$ are in some Q_i and if one of these arcs is in Q_i none of the remaining arcs involving u belongs to Q_j, $i \neq j$. The makespan is the cost of a *critical path* in G_S. A critical path is a longest cost path from node *start* to node *end*.

Figure 1 shows a solution graph for an instance with 3 jobs, 3 machines and 2 operators. Discontinuous arrows represent operator arcs. So, the sequences of operations assisted by operators O_1 and O_2 are $(\theta_{21}, \theta_{11}, \theta_{32}, \theta_{12}, \theta_{13})$ and $(\theta_{31}, \theta_{22}, \theta_{23}, \theta_{33})$ respectively. In order to simplify the picture, only the operator arc is drawn when there are two arcs between the same pair of nodes. Continuous arrows represent conjunctive arcs and doted arrows represent disjunctive arcs; in these cases only arcs not overlapping with operator arcs are drawn. In this example, the critical path is given by the sequence $(\theta_{21}, \theta_{11}, \theta_{32}, \theta_{12}, \theta_{33})$, so the makespan is 14.

In order to simplify expressions, we define the following notation for a feasible schedule. The *head* r_v of an operation v is the cost of the longest path from node *start* to node v, i.e. it is the value of st_v. The *tail* q_v is defined so as the value $q_v + p_v$ is the cost of the longest path from v to *end*. Hence, $r_v + p_v + q_v$ is the makespan if v is in a critical path, otherwise, it is a lower bound. PM_v and SM_v denote the predecessor and successor of v respectively on the machine sequence, PJ_v and SJ_v denote the predecessor and successor operations of v respectively on the job sequence and PO_v and SO_v denote the predecessor and successor operations of v respectively on the operator of v.

A partial schedule is given by a subgraph of G where some of the disjunctive and operator arcs are not fixed yet. In such a schedule, heads and tails can be estimated as

$$r_v = \max\{\max_{J \subseteq P(v)} \{\min_{j \in J} r_j + \sum_{j \in J} p_j\}, \tag{1}$$

$$\max_{J \subseteq PO(v)} \{\min_{j \in J} r_j + \sum_{j \in J} p_j\}, r_{PJ_v} + p_{PJ_v}\}$$

$$q_v = \max\{\max_{J \subseteq S(v)} \{\sum_{j \in J} p_j + \min_{j \in J} q_j\}, \tag{2}$$

$$\max_{J \subseteq SO(v)} \{\sum_{j \in J} p_j + \min_{j \in J} q_j\}, p_{SJ_v} + q_{SJ_v}\}$$

with $r_{start} = q_{end} = r_{O_i^{start}} = q_{O_i^{end}} = 0$ and where $P(v)$ denotes the disjunctive predecessors of v, so as for all $w \in P(v)$, $R_w = R_v$ and the disjunctive arc (w, v) is already fixed (analogously, $S(v)$ denotes the disjunctive successors of v). $PO(v)$ denotes the operator predecessors of v, i.e $w \in PO(v)$ if it is already established that $O_w = O_v$ and w is processed before v, so as the operator arc (w, v)

is fixed (analogously, $SO(v)$ are the operator successors of v).

Schedule Generation Schemes

In (Giffler and Thompson 1960), the authors proposed a schedule generation scheme for the $J||C_{max}$ problem termed $G\&T$ algorithm. This algorithm is an active schedule builder and it has been used in a variety of settings for job-shop scheduling problems. For example, in (Brucker, Jurisch, and Sievers 1994) it was used to devise a greedy algorithm, in (Bierwirth 1995) and (Mattfeld 1995) it was exploited as a decoder for genetic algorithms, and in (**?**) it was used to define the state space for a best-first search algorithm. Unfortunately, for other scheduling problems the original $G\&T$ algorithm is not longer suitable, but it has inspired the design of similar schedule builders such as the $EG\&T$ algorithm proposed in (Artigues, Lopez, and Ayache 2005) for the job-shop scheduling problems with sequence dependent setup times. In this section we propose a schedule generation scheme for the $JSO(n,p)$ which is also inspired in the $G\&T$ algorithm.

As it is done by the $G\&T$ algorithm, the operations are scheduled one at a time in sequential order within each job. When an operation u is scheduled, it is assigned a starting time st_u and an operator O_i, $1 \leq i \leq p$. Let SC be the set of scheduled operations at a given time and G_{SC} the partial solution graph built so far. For each operation u in SC, all operations in $P(u)$ or $PO(u)$ are scheduled as well. So, there is a path in G_{SC} from the node O_i^{start} to u through all operations in $PO(u)$ and a path from $start$ to u through all operations in $P(u)$. Let A be the set that includes the first unscheduled operation of each job that has at least one unscheduled operation, i.e.

$$A = \{v \notin SC, \nexists PJ_v \vee PJ_v \in SC\} \tag{3}$$

For each operation u in A, r_u is the starting time of u if u is selected to be scheduled next. In accordance with expression (1), r_u is greater or at least equal than both the completion time of PJ_u and the completion time of the last operation scheduled on the machine R_u. Moreover, the value of r_u also depends on the availability of operators at this time, hence r_v is not lower than the earliest time an operator is available. Let t_i, $1 \leq i \leq p$, be the time at which the operator i is available, then

$$r_u = \max\{r_{PJ_u} + p_{PJ_u}, r_v + p_v, \min_{1 \leq i \leq p} t_i\} \tag{4}$$

where v denotes the last operation scheduled having $R_v = R_u$. In general, a number of operations in A could be scheduled simultaneously at their current heads, however it is clear that not all of them could start processing at these times due to both capacity constraints and operators availability. So, a straightforward schedule generation scheme is obtained if each one of the operations in A is considered as candidate to be scheduled next.

If the selected operation is u, it is scheduled at the time $st_u = r_u$ and all the disjunctive arcs of the form (u, v), for all $v \notin P(u)$, get fixed. Operator arcs should also be fixed from the set of scheduled operations assisted by any of the operators. In principle, any operator i with $t_i \leq r_u$ can be selected for the operation u. So, if we start from the set A containing the first operation of each job and iterate until all the operations get scheduled, no matter what operation is selected in each step, we will finally have a feasible schedule, and there is a sequence of choices eventually leading to an optimal schedule, as it is established by the following result.

Proposition 1. *In at least one of the optimal schedules reachable from G_{SC} there is an operation $u \in A$ that is scheduled at a time $st_u = r_u^A$, where r_u^A is the head of u in G_{SC}.*

Proof. Let G_S be an optimal schedule reachable from G_{SC} and u^* be one of the operations in A such that none of the remaining operations in A is scheduled before u^* in G_S, i.e. $st_{u^*} \leq st_v$ for all $v \in A$. Then $P(u^*)$ is the same in G_S as it is in G_{SC}. If O_{u^*} is one of the operators that were available in G_{SC} at a time $t < r_{u^*}^A$, then, in G_S, $st_{u^*} = r_{u^*} = r_{u^*}^A$. Otherwise, i.e. if O_{u^*} is an operator such that in G_{SC} is available at a time $t > r_{u^*}^A$ then $st_{u^*} > r_{u^*}^A$. Let v be an operation in A assisted in G_S by an operator O_v, such that O_v is available at a time $t \leq r_{u^*}^A$ in G_{SC}, and O_v does not assist in G_S any other operation in A along the time interval $[t, st_v]$. As $st_v \geq st_{u^*}$ the operations u^* and $SO(u^*)$ can swap their assisting operator with operations v and $SO(v)$ and then u^* and v get scheduled at new starting times $st'_{u^*} = r_{u^*}^A$ and $st'_v = st_v$. This new schedule is not worse than G_S and so it is optimal. Therefore, in this optimal schedule one of the operations in A is scheduled at its head in G_{SC}. □

Let us now consider the operation v^* with the earliest completion time if it is selected from the set A, i.e.

$$v^* = \arg\min\{r_u + p_u; u \in A\} \tag{5}$$

and the set A' given by the operations in A that can start before the earliest completion of v^*, i.e.

$$A' = \{u \in A; r_u < r_{v^*} + p_{v^*}\} \tag{6}$$

If we restrict the selection, in each step of the previous scheduling algorithm, to an operation in A', at least one of the schedules that may be eventually reached is optimal. The reason for this is that for every operation u in A not considered in A', the possibility of being scheduled at a time $st_u = r_u^A$ remains in the next step. This is clear from the following result.

Proposition 2. *For every operation $u \in A \setminus A'$ the head r_u and the operators available to assist u are the same just before and just after operation v^* gets scheduled, if v^* is the first operation in A selected to be scheduled next.*

Proof. As $r_{v^*} + p_{v^*} \leq r_u^A$, for all $u \in A \setminus A'$, u could still be scheduled at time r_u^A after scheduling v^*, as at least the operator O_{v^*} would be available at this time. □

We can go further in restricting the choices in each step from the following observations. If the number of operators available is large enough, it is not necessary to take

all the operations in the set A' as candidate selections. Let $[T, C)$ be a time interval, where $C = r_{v^*} + p_{v^*}$ and $T = min\{r_u; u \in A'\}$, and the set of machines $R_{A'} = \{R_u; u \in A'\}$. If we consider the simplified situation where $r_u = T$, for all $u \in A'$ we can do the following reasoning: if, for instance, the number of machines in $R_{A'}$ is two and there are two or more operators available along $[T, C)$, then the set A' can be reduced to the operations requiring the machine R_{v*}. In other words, we can do the same as it is done in the $G\&T$ algorithm for the classical job-shop problem. The reason for this is that after selecting an operation v requiring R_{v*} to be scheduled, every operation $u \in A'$ requiring the other machine can still be scheduled at the same starting time as if it were scheduled before v, so as this machine may not be considered in the current step. However, if there is only one operator available along $[T, C)$ then A' may not be reduced, otherwise the operations removed from A' will no longer have the possibility of being processed at their current heads.

The reasoning above can be extended to the case where p' operators are available along $[T, C)$ and the number of machines in $R_{A'}$ is $m' \geq p'$. In this case A' can be reduced to maintain the operations of only $m' - p' + 1$ machines in order to guarantee that all the operations in A' have the opportunity to get scheduled at their heads in G_{SC}.

In general, the situation is more complex as the heads of operations in A' are distributed along the interval $[T, C)$ as well as the times at which the operators get available. Let $\tau_0 < \cdots < \tau_{p'}$ be the times given by the head of some operation in A' or the time at which some operator becomes available along the interval $[T, C)$. It is clear that $\tau_0 = T$ and $\tau_{p'} < C$. Let NO_{τ_i}, $i \leq p'$, be the number of operators available in the interval $[\tau_i, \tau_{i+1})$, with $\tau_{p'+1} = C$, R_{τ_i} the set of machines that are required before τ_{i+1},i.e $R_{\tau_i} = \{R_u; r_u \leq \tau_i\}$ and NR_{τ_i} be the number of machines in R_{τ_i}.

We now consider the time intervals from $[\tau_{p'}, C)$ backwards to $[T, \tau_1)$. If $NO_{\tau_i} \geq NR_{\tau_i}$ then only the operations of just one of the machines in R_{τ_i} should be maintained in A' due to the interval $[\tau_i, \tau_{i+1})$. Otherwise, i.e. if $NO_{\tau_i} < NR_{\tau_i}$, then the operations from at least $NR_{\tau_i} - NO_{\tau_i} + 1$ machines must be maintained from A' in order to guarantee that any operation requiring a machine in R_{τ_i} could be eventually processed along the interval $[\tau_i, \tau_{i+1})$. In other words, in this way we guarantee that any combination of NO_{τ_i} operations requiring different machines can be processed along the interval $[\tau_i, \tau_{i+1})$, as at least one operation requiring each of the $NR_{\tau_i} - NO_{\tau_i} + 1$ machines will appear in such combination of operations.

Here, it is important the following remark regarding the operator chosen for the operation u selected to be scheduled. As it was pointed, in principle any operator available at time r_u is suitable for u. However, we now have to be aware of leaving available operators for the operations that are removed from A'. In order to do that we use a strategy that consist in selecting an operator i available at the latest time t_i such that $t_i \leq r_u$. In this way, we maximize the availability of operators for the operations to be scheduled in the next steps. To be more precise, we guarantee that any operation removed from A' can be scheduled at its current head in a subsequent step.

From the interval $[\tau_{p'}, C)$ at least the operation v^* is maintained and then from the remaining ones some new operations are added. The set of operations obtained in this way is termed B and it is clear that $|B| \leq |A'| \leq |A|$. An important property of this schedule generation scheme is that if the number of operators is large enough, in particular if $p \geq \min(n, m)$ so as $JSO(n, p)$ becomes $J||C_{max}$, it is equivalent to the $G\&T$ algorithm. So, we call this new algorithm $OG\&T$ (Operators $G\&T$). We will distinguish three variants of this algorithm defined by the sets A, A' and B of candidate operations, termed $OG\&T_A$, $OG\&T_{A'}$ and $OG\&T_B$ (or simply $OG\&T$) respectively. From the reasoning above the following result can be established.

Theorem 1. *Let $\mathcal{P}$ be a $JSO(n, p)$ instance. The set $\mathcal{S}$ of schedules that can be obtained by the $OG\&T$ algorithm to $\mathcal{P}$ is dominant, i.e. $\mathcal{S}$ contains at least one optimal schedule.*

Search Algorithm

Our approach is an implementation of the best-first search strategy proposed by Nilsson (Hart, Nilsson, and Raphael 1968), (Pearl 1984). This algorithm starts from an initial state and then in each step it expands the first one of the set of candidate states stored in the $OPEN$ list. The $OPEN$ list is sorted by non decreasing $f-$values, where f is the evaluation function for the states. For a state s, $f(s)$ gives an estimation of the cost of the best schedule that can be reached from s, denoted as $f^*(s)$.

In the following four subsections we describe the main components of the best-first search algorithm: the search space, the heuristic estimations, a method to prune the search tree based on dominance relations among states and finally a method to obtain upper bounds along the search.

Search Space

The search space is derived from the $OG\&T$ schedule generation scheme. For a problem instance $\mathcal{P}$, in the initial state, none of the operations are scheduled yet and so it is defined by the disjunctive graph G. In intermediate states, a subset of operations SC are already scheduled, i.e. each operation $v \in SC$ has been assigned a starting time st_v and an assisting operator O_v. The set SC is built so as for all $v \in SC$ PJ_v is in SC as well if this operation exists. A goal state is defined by a solution graph. To obtain the successors of a state defined by G_{SC}, the set B (or A or A') is built as it was indicated previously and then one successor state is generated from each operation u in B (or A or A') in which u is scheduled at its current head in G_{SC}.

We consider this search space as a tree, even though it could be considered as a graph as well. This is clear as two intermediate states with the same operations scheduled might represent the same subproblem and so this situation could be checked for avoiding duplications. However, as we will see below, the test defined to establish dominance relations among states is in fact a generalization of the checking for duplications.

From theorem 1 above, it is clear that the search tree includes some optimal solution.

Heuristic Functions

We use the A^* version of best-first search, so the evaluation function is of the form $f(s) = g(s) + h(s)$, where $g(s)$ denotes the makespan accumulated in the state s, i.e. the maximum completion time of the operations scheduled in s. Then, $h(s)$ is a heuristic function that estimates the additional makespan required to reach a solution from s. We consider two admissible heuristics derived from problem relaxations.

The first one, termed h_{JPS}, is taken from the classical job-shop scheduling problem, so the operators constraints are relaxed. Additionally, the capacity constraints of all machines except those of one machine R_i are relaxed to obtain an instance of the One Machine Sequencing (OMS) problem with heads and tails. This version of the OMS problem is still NP-hard, but if the non-preemption constraint for the operations requiring R_i is also relaxed, the resulting simplified problem is solvable in polinomial time.

The optimal preemptive schedule can be computed by the algorithm proposed in (Carlier 1982) in $O(k \times \log k)$, where k is the number of unscheduled operations requiring R_i. The same procedure is applied to all machines with some unscheduled operations, and the value of $h_{JPS}(s)$ is obtained from the maximum makespan for all these preemptive schedules, C_{max}^{pmpt}. This is an estimation for $f(s)$, so $h_{JPS}(s) = \max(0, C_{max}^{pmpt} - g(s))$. A quite similar heuristic has been used to solve the problem $J||\Sigma C_i$ in (**?**).

We propose here a second admissible heuristic which is based on relaxing the capacity constraints for all the machines, the non preemption constraints for all the operations and the precedence constraints (even within an operation, so as two or more of its time units may be processed in parallel). In this way, we only maintain the operator constraints and the lower bounds for the starting times due to the heads of the operations.

The resulting heuristic is termed h_{OP} and it is computed as follows: for every t from t_{min} to $t_{max} - 1$, where $t_{min} = \min\{r_u; u \notin SC\}$, and $t_{max} = \max\{r_u + p_u; u \notin SC\}$, calculate x_t as the number of operators available in the time interval $[t, t+1)$ and y_t as the number of unscheduled operations u such that $r_u \leq t$ and $r_u + p_u \geq t + 1$. The number of time units processed in the solution to the relaxed problem, along the interval $[t, t+1)$, is computed as $\min(y_t + z_t, x_t)$, where z_t denotes the remaining time units from the previous interval (with $z_{t_{min}} = 0$). Then $z_{t+1} = \max((y_t + z_t) - x_t, 0)$. The final $z_{t_{max}}$ time units can be uniformly distributed among all the p operators and so the completion time is calculated as $t_{max} + \lceil z_{t_{max}}/p \rceil$. As before, this value is an estimation for $f(s)$, so $h_{OP}(s) = \max(0, t_{max} + \lceil z_{t_{max}}/p \rceil - g(n))$.

Finally we take $h(s) = \max(h_{JPS}(s), h_{OP}(s))$.

Dominance Rules

The effective search tree may be reduced by means of dominance relations among states similar to that exploited in (**?**) for the classic job-shop problem. Given two search states s_1 and s_2, s_1 dominates s_2 iff $f^*(s_1) \leq f^*(s_2)$. In general, dominance relations cannot be easily established, but in some particular cases an efficient condition for dominance can be defined. For the search space above, a simple and effective dominance rule is defined as follows. If s_1 and s_2 are states having the same operations scheduled, SC, then s_1 dominates s_2 if the following two conditions hold, where $r_v(s)$ denotes the head of v in state s and $av(s)$ the availability of operators in this state:

1. $r_v(s_1) \leq r_v(s_2)$, for all $v \notin SC$.
2. $av(s_1) \geq av(s_2)$.

From condition (1.) it is clear that the makespan of the best schedule that can be obtained from s_1 is not greater than the makespan of the best schedule reachable from s_2, provided that the availability of operators in s_1 is not worse than it is in s_2. The availability of operators in a state can be evaluated as follows. Let $t_1 \leq \cdots \leq t_p$ the times at which the operators get idle in the state s, if u^* is the unscheduled operation with the lowest head in s, then none of the operators can get busy again before r_{u^*}, so we can consider that the operators are actually available for the unscheduled operations at times $t'_1 \leq \cdots \leq t'_p$, where $t'_i = max(r_{u^*}, t_i)$. So the availability of operators in state s is defined as the ordered vector $av(s) = (t'_1, \ldots, t'_p)$ and $av(s_1) \leq av(s_2)$ iff $t'_{1i} \leq t'_{2i}, 1 \leq i \leq p$.

The implementation of the dominance rule can be done as follows. When a state s is taken for expansion, s is compared to all the expanded states having the same operations scheduled. This can be done efficiently if the expanded states are stored in a CLOSED list implemented as a hash table where the key values are bit-vectors representing the scheduled operations. Moreover, this rule may be improved from the following result that establishes a sufficient condition for the conditions (1.) and (2.) not to hold.

Proposition 3. *If the heuristic estimation is obtained from a problem relaxation, i.e. $f(s)$ is the cost of an optimal solution to the relaxed problem obtained from s in accordance with that problem relaxation, and the states s_1 and s_2 fulfill both conditions (1.) and (2.), then $f(s_1) \leq f(s_2)$.*

Proof. It is trivial, as any solution to the relaxed instance obtained from s_2 is a solution to the relaxed instance obtained from s_1. □

So, when a state s is expanded, it has only to be matched to states s' in CLOSED having the same operations scheduled ($f(s') \leq f(s)$ due to h being monotonic). Furthermore, s may also be matched with states s' in OPEN such that $f(s) = f(s')$ in order to verify whether s' dominates s, so as s is pruned before expanding.

As both heuristics h_{JPS} and h_{OP} are obtained from problem relaxations, the evaluation functions defined as $f_{JPS}(s) = g(s) + h_{JPS}(s)$ and $f_{OP}(s) = g(s) + h_{OP}(s)$ fulfill the condition of proposition (3). As $f(s) = \max(f_{JPS}(s), f_{OP}(s))$, the condition may be evaluated on f, due to the fact that $f(s_1) > f(s_2)$ implies that at least one of the conditions $f_{JPS}(s_1) > f_{JPS}(s_2)$ or $f_{OP}(s_1) > f_{OP}(s_2)$ holds.

As we have commented above, this pruning method generalizes the checking for duplication as in these situations the two nodes dominate each one to the other.

Upper Bounds Calculation

Best-first search can be combined with a greedy algorithm to obtain an any-time algorithm in the following way: each time a state s is selected for expansion, the greedy algorithm is issued from s to obtain one of the solutions reachable from s. This allows the algorithm to obtain approximate solutions from the beginning of the search and a number of candidate states to be pruned under the condition $f(s) \geq UB$, where UB is the best upper bound found so far. Of course, this process is time consuming so as its effectiveness must be evaluated by experimentation. One possibility to reduce the time taken is not to issue the greedy algorithm from every expanded state but just once from every 100 expansions or so.

We have chosen to introduce upper bounds calculation in the following way: starting from the selected state s, the greedy algorithm iterates until a goal state is reached. In each iteration the set of candidate operations B is computed as done by the $OG\&T$ algorithm, then an operation is selected from B to be scheduled, in accordance with a heuristic estimation. In the experimental study we have opted to use the $h-$values for this purpose, even though a less time consuming estimation could be used instead.

Computational Results

The purpose of the experimental study is to assess our approach and to compare it with the state-of-the-art methods, namely the dynamic programming and the heuristic algorithms given in (Agnetis et al. 2010). We have experimented across two sets of instances. The first one includes a number of instances from the OR-library, in particular $FT06, 10, 20$, $LA01-20$, $ABZ5, 6$ and $ORB01-10$. For each instance, all values in the interval $[1, \min(n, m)]$ are considered as the number of operators p. The second benchmark is that proposed in (Agnetis et al. 2010), all these instances have $n = 3$ and $p = 2$ and are characterized by the number of machines (m), the number of operations per job (v_{max}) and the range of processing times (p_i). A set of small instances was generated combining three values of each parameter: $m = 3, 5, 7$; $v_{max} = 5, 7, 10$ and $p_i = [1, 10], [1, 50], [1, 100]$. Also, a set of larger instances was generated with $m = 3$, combining $v_{max} = 20, 25, 30$ and $p_i = [1, 50], [1, 100], [1, 200]$. In all cases, 10 instances were considered from each combination. The sets of small instances are identified by numbers from 1 to 27: set 1 corresponds to triplet $3-5-10$, the second is $3-5-50$ and so on. The sets of large instances are identified analogously by labels from $L1$ to $L9$.

In all the experiments we have given a time limit of 3600 s, so the algorithms finish after this time, or if the memory gets exhausted (limit of 23,3 GB), before an optimal solution is reached. The target machine was Intel Xeon (2,26 GHz), 24 GB RAM. and the algorithms are coded in C++.

In the first series of experiments we tried to assess each one of the elements of the proposed algorithm. Firstly we evaluated the improvement produced by the reduction of the set of candidate operations from A to A' and finally to B. So we have experimented with the three variants $OG\&T_A$, $OG\&T_{A'}$ and $OG\&T_B$. Then we evaluated the capability of the pruning rules to reduce the search space. Finally, we evaluated the effectiveness of combining best-first with greedy search; in these experiments the greedy algorithm was issued once every 100 expansions. To do that we have experimented with 6 instances build from the instance $FT06$ (6 jobs and 6 machines), p ranging from 1 to 6, and guiding the search with the heuristic h. The results are summarized in Table 1. For each combination, we report the average number of nodes expanded ($\#Exp.$), the average time taken in seconds ($T.(s)$), the number of instances solved to optimality ($\#Sol.$) and the maximum memory consumed in MB ($RAM(MB)$).

As we can observe, not all combinations are able to solve all the 6 instances by the time limit. It is also clear that each one of the elements evaluated contributes to improving the performance of the algorithm and that the synergy gained from their use in combination makes the algorithm much more efficient and effective. It is remarkable, for example, the exponential reduction in the number of expansions produced by the pruning rules in combination with algorithms $OG\&T_A$ and $OG\&T_{A'}$. These rules are also effective in combination with $OG\&T_B$, as they allow the algorithm to reduce the number of expanded nodes by about 50%. The greedy algorithm is worth using as well. For example, in combination with $OG\&T_{A'}$ it allows the algorithm to solve all the instances without using the pruning rules, and even in combination with $OG\&T_B$ it reduces the number of expanded nodes. So, we consider all these elements in the remaining of the experimental study.

In the second series of experiments, we have evaluated the heuristics h_{JPS} and h_{OP} separately and in combination. We have also used the same instances from $FT06$ and the results are summarized in Figure 2. Here we report the number of expanded nodes (in a logarithmic scale with base 10) for each combination of instance and heuristic. The most relevant observation in this case is the com-

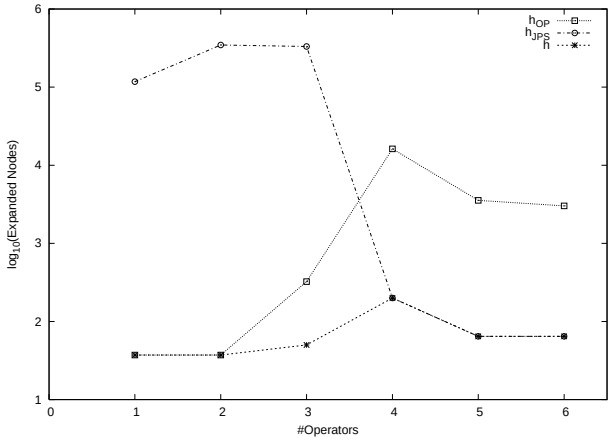

Figure 2: Heuristics evaluation for instance FT06.

Table 1: Summary of results from instance FT06.

		no greedy search			greedy search		
		A	A'	B	A	A'	B
No Pruning	#Exp	1262386	2229326	152	2131256	618043	127
	T.(s)	403	430	0	725	157	0
	#Sol.	2/6	3/6	6/6	2/6	6/6	6/6
	RAM(MB)	23859	23859	5	23862	9287	5
Pruning	#Exp	1955	237	76	1460	187	58
	T.(s)	1	0	0	1	0	0
	#Sol.	6/6	6/6	6/6	6/6	6/6	6/6
	RAM(MB)	51	6	3	20	5	3

Table 2: Summary of results from instances with 5 machines.

	$10jobs(LA01-05)$				$15jobs(LA06-10)$				$20jobs(LA11-15, FT20)$			
#Op.	#Exp.	T.(s)	#Sol.	%Err.	#Exp.	T.(s)	#Sol.	%Err.	#Exp.	T.(s)	#Sol.	%Err.
1	0	0	5/5	0,0	0	0	5/5	0,0	0	0	6/6	0,0
2	119	0	5/5	0,0	275	1	5/5	0,0	202	0	6/6	0,0
3	6434	2	5/5	0,0	13871	7	5/5	0,0	320557	390	5/6	0,1
4	799480	809	4/5	0,1	173500	112	5/5	0,0	519160	558	4/6	1,3
5	4093	1	5/5	0,0	561	0	5/5	0,0	8327	5	6/6	0,0

plementarity of heuristics h_{JPS} and h_{OP}. This is quite reasonable as h_{OP} is expected to be a better estimation than h_{JPS} when the number of operators is small and the contrary can be expected when the number of operators is large. So, when they are used in combination to define $h(s) = \max(h_{JPS}(s), h_{OP}(s))$, a synergetic effect is obtained as well. Also, it can be observed that the hardest instances are those with intermediate values of p. As we will see in the next experiments, this happens for larger instances too. We will use h in all the remaining experiments.

In the next experiments we have considered instances larger than $FT06$ from the OR-library. Firstly, instances with $m = 5$ and different number of jobs with 5 operations each: $LA01-05$ with $n = 10$, $LA06-10$ with $n = 15$ and $LA11-15$ and $FT20$ with $n = 20$. And then instances with 10 jobs and 10 machines: $FT10$, $LA16-20$, $ABZ5, 6$ and $ORB01-10$. The results of these experiments are reported in Tables 2 and 3 respectively. From Table 2 we can observe that the hardest instances are those with 3 and 4 operators. All but 3 of the 16 instances get solved before the memory gets exhausted, and for the 3 unsolved instances the error in percent (w.r.t. the best lower bound given by the f-value of the last node expanded) is very low ($1,3\%$ in the worst case). So, these instances are within the size limit that the algorithm can solve to optimality. Even though the branching factor might be really high (20 in the worst case), the pruning rules and the heuristic h are able to reduce the effective search space thus making the algorithm efficient.

On the other hand, Table 3 shows that instances with 10 jobs and 10 machines are really hard to solve. In this case the heuristic estimation is less accurate and so many of the instances (mainly the hardest ones with an intermediate number of operators) cannot be solved to optimality by the time or memory limits. However, the algorithm reaches very good upper bounds in all cases thanks to using the greedy algorithm.

Table 3: Summary of results from instances with 10 jobs and 10 machines.

#Op.	#Exp.	T.(s)	#Sol.	%Err.
1	0	0	18/18	0,0
2	471	0	18/18	0,0
3	260537	478	15/18	0,2
4	605993	1074	11/18	0,7
5	931227	2078	3/18	3,3
6	993524	2708	0/18	6,4
7	1647244	3149	0/18	4,2
8	1864672	2108	9/18	1,1
9	1753181	1839	10/18	0,8
10	1737957	1788	9/18	0,9

In the last serie of experiments, we have considered the 360 instances proposed in (Agnetis et al. 2010). Table 4 reports the results of these experiments together with the results from the best exact and approximate algorithms given in (Agnetis et al. 2010), namely the dynamic programming algorithm (DP) and the heuristic algorithms ($Heur$ and $Heur$+). The results are averaged for subsets of instances with the same number of operations per job v_{max}. For the best-first algorithm (BF), we report the maximum memory requirements in addition to the average time taken and the average number of expanded nodes. All these instances are optimally solved by BF, however some of them are no solved by DF after 3 hours. These instances belong to the sets indicated with the symbol * in Table 4. As we

Table 4: Summary of results from instances with 3 jobs and 2 operators.

Instances		BF			T.(s)				
		#Exp.	RAM(MB)	T.(s)	DP	Heur	Heur+	Heur Gap	Heur+ Gap
SMALL	1-9	38	2,0	0,00	0,03	0,01	0,04	15,33	11,22
	10-18	94	4,1	0,02	1,29	0,04	0,47	14,81	10,98
	19-27	160	4,4	0,01	14,93	0,47	4,86	15,31	11,12
LARGE	L1-L3	5184	125,1	2,07	654,10	9,40	91,47	17,80	12,90
	L4-L6	10891	353,6	5,17	1683,60*	35,77	366,73	15,40	11,97
	L6-L9	18206	808,0	10,53	4351,00*	97,00	823,97	16,50	11,57

can see, the time taken by BF is about 2 orders of magnitude lower than the time taken by DP (considering 3 hours for the unsolved instances). This difference is really significant in spite of the differences in the target machine (DP has been run on a PC with a 3 GHz clock, 2 GB RAM and coded in C++). Also, the time taken by BF is much lower than the time taken by $Heur$ and $Heur$+ to obtain suboptimal solutions with an error in percent in a range about 10 to 20. The memory requirement of BF is 808 MB in the worst case.

From this experimental study we can conclude that the proposed best-first search algorithm outperforms the the exact and approximate algorithms proposed in (Agnetis et al. 2010). It can solve instances with 20 jobs with 5 operations each and 5 machines to optimality; and for instances with 10 jobs with 10 operations each and 10 machines, it can reach suboptimal solutions with a very low error. So, at difference of the algorithms proposed in (Agnetis et al. 2010), the proposed best-first search algorithm can be efficiently scaled to solve instances with more than 3 jobs and more than 2 operators.

Conclusions

We have proposed an algorithm that combines best-first search and greedy search to solve the job-shop scheduling problem with operators. This algorithm is based on a new schedule generation scheme termed $OG\&T$. The effectiveness of this algorithm relies on two heuristic estimations derived from problem relaxations and a method for pruning states based on dominance relations among states. We have reported results from an experimental study across conventional and new instances. This study shows that our approach outperforms the best-known exact and approximate approaches, that as far as we know are those proposed in (Agnetis et al. 2010).

As future work we plan to experiment with heuristic search algorithms other than best-first, for example partially informed depth-first search as it was done in (**?**) for the classical job-shop problem, and to consider other variants of the problem more interesting from a practical point of view, focused on new operator constraints due to differences in the skills or time constraints due to labor rules. Also, it would be interesting to relax the admissibility of heuristic estimations and dominance rules in order to obtain good approximations for large instances.

Acknowledgments

We would like to thank Andrea Pacifici and Marta Flamini for making their benchmark instances available.

References

Agnetis, A.; Flamini, M.; Nicosia, G.; and Pacifici, A. 2010. A job-shop problem with one additional resource type. *Journal of Scheduling* (DOI 10.1007/s10951-010-0162-4).

Artigues, C.; Lopez, P.; and Ayache, P. 2005. Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. *Annals of Operations Research* 138:21–52.

Bierwirth, C. 1995. A generalized permutation approach to jobshop scheduling with genetic algorithms. *OR Spectrum* 17:87–92.

Brucker, P.; Jurisch, B.; and Sievers, B. 1994. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics* 49:107–127.

Carlier, J. 1982. The one-machine sequencing problem. *European Journal of Operational Research* 11:42–47.

Giffler, B., and Thompson, G. L. 1960. Algorithms for solving production scheduling problems. *Operations Research* 8:487–503.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Sys. Science and Cybernetics* 4(2):100–107.

Mattfeld, D. C. 1995. *Evolutionary Search and the Job Shop Investigations on Genetic Algorithms for Production Scheduling*. Springer-Verlag.

Pearl, J. 1984. *Heuristics: Intelligent Search strategies for Computer Problem Solving*. Addison-Wesley.