Annals of Operations Research 131, 215–235, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

A Rollout Metaheuristic for Job Shop Scheduling Problems

CARLO MELONI
Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, via E. Orabona, 4, 70125 Bari, Italy

DARIO PACCIARELLI[*] and MARCO PRANZO pacciarelli@dia.uniroma3.it
Dipartimento di Informatica e Automazione, Università "Roma Tre", via della Vasca Navale 79, 00146 Roma, Italy

Abstract. In this paper we deal with solution algorithms for a general formulation of the job shop problem, called alternative graph. We study in particular the job shop scheduling problem with blocking and/or no-wait constraints. Most of the key properties developed for solving the job shop problem with infinite capacity buffer do not hold in the more general alternative graph model. In this paper we report on an extensive study on the applicability of a metaheuristic approach, called rollout or pilot method. Its basic idea is a look-ahead strategy, guided by one or more subheuristics, called pilot heuristics. Our results indicate that this method is competitive and very promising for solving complex scheduling problems.

Keywords: metaheuristics, job shop scheduling, rollout, alternative graph

1. Introduction

The job shop scheduling problem is the problem of allocating machines to competing jobs over time, subject to the constraint that each machine can handle at most one job at a time. It has been approached from a variety of perspectives, using formulations and tools of various disciplines, such as control theory (Cassandras, Lafortune, and Olsder, 1995), combinatorial optimization (Adams, Balas, and Zawack, 1988; Carlier and Pinson, 1994), and artificial intelligence (Sadeh, Sycara, and Xiong, 1995; Nuijten and Le Pape, 1998). These different approaches resulted in several solution techniques, combining generic as well as problem specific strategies, and reflecting the area from which this optimization problem is derived.

From the point of view of the modeling techniques, most of these works are based on the disjunctive graph formulation of Roy and Sussman (1964). The disjunctive graph model has been extensively studied in order to develop efficient solution algorithms and to solve problems from industrial practice. To this second aim, several authors observed that the disjunctive graph can be easily adapted to deal with many practical issues (see, for example, White and Rogers, 1990; Schutten, 1998). A strong limitation that still remains in these models is that they disregard the capacity of intermediate buffers. In fact,

[*] Corresponding author.

in many practical situations buffer capacity has to be taken into account, at least at some stages of the production process, as shown, e.g., in (Hall and Sriskandarajah, 1996). In scheduling theory the absence of a buffer is often modeled with blocking or no-wait constraints. In particular, a blocking constraint models the absence of storage capacity between machines, while a no-wait constraint occurs when two consecutive operations in a job must be processed without any interruption. To incorporate this restriction the disjunctive graph formulation can be adapted to a more general model called *alternative graph* (Mascis and Pacciarelli, 2000). In this paper we deal with solution techniques for a general alternative graph formulation of the job shop problem, and report in particular on the results for the job shop scheduling problem with blocking and/or no-wait constraints.

As solution technique, we discuss and apply a metaheuristic strategy known as the *rollout* method (Bertsekas, Tsitsiklis, and Wu, 1997) or the *pilot* method (Duin and Voß, 1999). This is a quite new methodology, independently developed for the approximate solution of discrete optimization problems by Bertsekas, Tsitsiklis, and Wu (1997) and Duin and Voß (1999).

With this metaheuristic method, the solution s of an optimization problem is considered as a collection of m components, e.g., a value for each decision variable, or $s = (s_1, s_2, \dots, s_{m-1}, s_m)$. The problem is then solved sequentially by choosing the components $s_1, s_2, \dots, s_{m-1}, s_m$ one at a time. This iterative process is guided by one or more subheuristics, called pilot heuristics.

The main advantage of the rollout and pilot metaheuristics is that they are not based on specific properties of the problem. Rather they are based on generic solution concepts that guide underlying heuristic solution procedures. These underlying procedures may incorporate problem specific knowledge (but need not). This is particularly useful when dealing with very general problems and models that, as in the case under consideration, suffer for a lack of specific properties to be used in the design of efficient solution algorithms.

The paper is organized as follows. In section 2 a brief survey on metaheuristics and rollout techniques is reported, in section 3 we introduce the notation and the alternative graph formulation. In section 4 we discuss greedy heuristic algorithms for solving a general scheduling problem, modeled by means of the alternative graph formulation. In section 5 we discuss our rollout scheme and report on our computational experiments. Some conclusions follow.

2. Literature review

A considerable effort has been devoted in the past forty years to design heuristic procedures for solving the job shop scheduling problem with infinite capacity buffers (in the following Ideal Job Shop Problem, or IJSP). Most of these procedures are greedy algorithms, based on list schedules associated with dispatching rules. As reported by several authors (e.g., Lawrence, 1984; Pinson, 1997), the behaviour of such an approach is quite erratic.

A common approach for the IJSP consists of repeatedly enlarging a consistent partial schedule. Besides the rollout method, most of the greedy heuristics are based on this idea, as well as many other sophisticated algorithms. The shifting bottleneck algorithm of Adams, Balas, and Zawack (1988) falls in the latter category. The algorithm sequences, at each iteration, all the operations to be performed on the same machine. Among the earliest papers closely related to the rollout method, we cite the beam search technique, first used in artificial intelligence for the speech recognition problem (Lowerre, 1976), and then used by Fox (1983) for solving complex scheduling problems.

A detailed survey on solutions techniques for IJSP is provided by Błażewicz, Domschke, and Pesch (1996). In view of their extensive review, including even efficient branch and bound procedures, we provide only a brief summary here focusing on heuristics. The most significant results have been reported by using metaheuristic solution procedures. A metaheuristic is an iterative solution procedure, which combines some subordinate heuristic tools into a more sophisticated framework (Glover, 1986). A master process guides and modifies the operations of these heuristics in order to efficiently produce high-quality solutions. It may manipulate a complete or partial single solution, or a collection of solutions at each iteration. Also the subordinate heuristics can be constructive, enumerative or iterative procedures or hybrids. Such methods cover a large family of techniques, varying from rather simple to very sophisticated (see, e.g., (Voß et al., 1999)). The family of metaheuristics includes probabilistic or deterministic learning methods. Among the techniques that have been applied to IJSP, genetic algorithms (Dorndorf and Pesch, 1995), randomized adaptive search and simulated annealing fall in the first category, while methods based on the shifting bottleneck procedure and tabu search fall in the second one. According to the literature on IJSP, from the computational experiences of various authors, shifting bottleneck procedures and tabu search approaches provide better strategies than probabilistic methods, both in terms of computing time and quality of the solutions found (Jain and Meeran, 1999; Pinson, 1997; Vaessens, Aarts, and Lenstra, 1992). In particular, from a comparison between the shifting bottleneck procedure and tabu search techniques, the reiterated guided local search algorithm based on the shifting bottleneck method by Balas and Vazacopoulos (1998) outperforms the other methods in terms of solution quality, while the tabu search based algorithm by Nowicki and Smutnicki (1996) performs best in terms of computational time.

A drawback of most metaheuristic methods is that they strongly depend on particular combinatorial properties of the problem for which they are developed. Clearly, mathematical properties help in order to obtain high quality solutions within a short computing time, but in many cases the solution procedures based on these properties are much less effective, or unusable, when applied to more general problems arising from industrial practice. In particular for the job shop problem, most of the techniques which are proven to be effective for IJSP are based on several properties that do not hold for more constrained cases.

A recent research line within the context of metaheuristics aims at developing general frameworks easily adaptable to deal with many different problems. Such methods are not based on strong properties of the problem, but the search process is guided by one or more heuristic procedures. In this paper we apply one such technique, called rollout, to the context of general job shop problems. Rollout algorithms are a class of metaheuristic methods inspired by the methodology of neuro-dynamic programming (Bertsekas and Tsitsiklis, 1996). They were first used for solving discrete optimization problems by Bertsekas, Tsitsiklis, and Wu (1997). The resulting algorithm is very similar to the pilot method of Duin and Voß (1999).

The main idea of a rollout algorithm is to employ one or more fast algorithms for the construction of a good heuristic solution. We next illustrate the general scheme of this method.

Given an optimization problem P, let S be the finite set of feasible solutions, and $c(s)$ be a cost function, for each $s \in S$. Solving P consists of finding a solution $s \in S$ that minimizes $c(s)$. Each solution s can be considered as a collection of m components, e.g., a value for each decision variable, or $s = (s_1, s_2, \ldots, s_{m-1}, s_m)$. In the rollout scheme, a master process produces a solution sequentially, by fixing the components $s_1, s_2, \ldots, s_{m-1}, s_m$ one at a time.

A partial solution in which the value of k components is fixed is called a *k-state* S_k. For example $S_k = (s_1, s_2, \ldots, s_{k-1}, s_k)$ consists of the first k components of a solution. The 0-state is a dummy state, corresponding to the situation in which no component has been fixed yet. From each (k)-state it is possible to move to a $(k + 1)$-state by fixing a new component not already fixed.

At the kth iteration of the rollout metaheuristic the master process evaluates a heuristic function or a *scoring function* for each (seemingly) feasible component which can be added to the current state S_{k-1} and chooses the most promising component, i.e. the one having the smallest scoring function. The iteration is repeated until a complete solution is found.

The basic idea for computing the scoring functions at each iteration is a look-ahead strategy based on heuristic algorithms. Given S_k, the scoring function $H(S_k)$ denotes the cost of the solutions found by the subheuristics, starting from S_k. We set $H(S_k) = +\infty$ if the heuristics do not succeed in finding a feasible solution from S_k. To this aim, one or more subheuristics are needed, able to compute a complete solution starting from a given k-state S_k.

Two approaches suggested in (Bertsekas, Tsitsiklis, and Wu, 1997) consist of computing the scoring function at each state as:

- the weighted sum of the values obtained by the heuristic procedures,
- the minimum value obtained by the heuristic procedures.

In this paper we will consider only one subheuristic at the time, as in (Duin and Voß, 1999), and we aim at exploring the dependence of the quality of the final solution from different subheuristics.

3. Notation and problem definition

In this section we introduce the notation and a more precise specification of the problem under consideration. In the usual definition of the job shop problem a job must be processed on a set of machines. The sequence of machines for each job is prescribed, the processing of a job on a machine is called an *operation* and cannot be interrupted.

In this paper we focus on the sequencing of operations rather than jobs. We have therefore a set of operations $N = \{o_0, o_1, \dots, o_n\}$ to be performed on a set of machines. Each operation o_i requires a specified amount of processing p_i on a specified machine $M(i)$, and cannot be interrupted from its starting time t_i to its completion time $c_i = t_i + p_i$. o_0 and o_n are dummy operations, with zero processing time, that we call *start* and *finish*, respectively. Each machine can process only one operation at a time.

There is a set of precedence relations among operations. A *precedence relation* (i, j) is a constraint on the starting time of operation o_j, with respect to t_i. More precisely, the starting time of the successor o_j must be greater or equal to the starting time of the predecessor o_i plus a given *delay* f_{ij}, which in our model can be either positive, null or negative. A positive delay may represent, for example, the fact that operation o_j may start processing only after the completion of o_i, plus a possible setup time. A delay smaller or equal to zero represents synchronization between the starting times of the two operations. Finally, we assume that o_0 precedes $o_1, \dots, o_n$, and o_n follows $o_0, \dots, o_{n-1}$.

Precedence relations are divided into two sets: *fixed* and *alternative*. Alternative precedence relations are partitioned into m pairs. They usually represent the constraints that each machine can process only one operation at a time.

A *schedule* is an assignment of starting times $t_0, t_1, \dots, t_n$ to operations $o_0, o_1, \dots, o_n$, respectively, such that all fixed precedence relations, and exactly one for each pair of the alternative precedence relations, are satisfied. W.l.o.g. we assume $t_0 = 0$. The goal is to minimize the starting time of operation o_n. Therefore, this problem can be formulated as a particular *disjunctive program*, i.e. a linear program with logical conditions involving operations *and* ($\wedge$, conjunction) and *or* ($\vee$, disjunction), as in (Balas, 1979).

Problem 3.1.

$$\begin{aligned}
&\min\ t_n \\
&\text{s.t.}\ \ t_j - t_i \geqslant f_{ij} && (i, j) \in F, \\
&\qquad (t_j - t_i \geqslant a_{ij}) \vee (t_k - t_h \geqslant a_{hk}) && \big((i, j), (h, k)\big) \in A.
\end{aligned}$$

Associating a node to each operation, problem 3.1 can be usefully represented by the triple $G = (N, F, A)$ that we call an *alternative graph*. The alternative graph is as follows. There is a set of nodes N, a set of directed arcs F and a set of m pairs of directed arcs A. Arcs in the set F are *fixed* and f_{ij} is the length of arc $(i, j) \in F$. Arcs in the set A are *alternative*, and a_{ij} is the length of the alternative arc (i, j). If $((i, j), (h, k)) \in A$, we say that (i, j) is *the alternative* of (h, k). In our model the arc lengths can be either positive, null or negative. In our definition, an alternative arc belongs to one pair only. In other words, if we have two pairs $((i, j), (h, k))$ and $((i, j), (u, v)) \in A$, we consider the two occurrences of (i, j) as two different arcs. With this notation, a solution is associated to each choice of m alternative precedence relations, one from each pair. The solution is feasible if there are no positive length cycles in the resulting graph.

With the notation introduced in the previous section, for a given alternative graph $G = (N, F, A)$, a k-state S_k is a set of k arcs obtained from A by choosing at most one arc from each pair. The state is a *solution* if exactly m arcs are chosen. Given a pair of alternative arcs $((i, j), (h, k)) \in A$, we say that (i, j) is *selected* in S_k if $(i, j) \in S_k$, whereas we say that (i, j) is *forbidden* in S_k if $(h, k) \in S_k$. Finally, the pair is *unselected* if neither (i, j) nor (h, k) is selected in S_k.

Given a k-state S_k let $G(S_k)$ indicate the graph $(N, F \cup S_k)$. We say that a k-state S_k is *feasible* if the graph $G(S_k)$ has no positive length cycles. Given a feasible k-state S_k, we call an *extension* of S_k a feasible solution S_m such that $S_k \subseteq S_m$, if it exists. The value obtained by the scoring function in S_k is the length of a longest path from node 0 to node n in an extension of $G(S_k)$.

The alternative graph is a generalization of the disjunctive graph of Roy and Sussman (1964). In fact, in the disjunctive graph the pairs of alternative arcs (called disjunctive arcs) are all in the form $(i, j), (j, i)$, where i and j are two operations to be processed on the same machine.

In the remaining part of this section we briefly illustrate the alternative graph model of some typical constraints arising in scheduling problems.

Let us first consider a pair of consecutive operations in a job, say o_i and o_j, and assume that o_j must start processing within k time units after the completion of o_i ($k \geqslant 0$). We call it the *perishability* constraint, since it represents, for example, the fact that a job deteriorates when stored for more than k time units between the two consecutive operations. We can represent this constraint with a pair of fixed arcs (i, j) and (j, i) in F of lengths $f_{ij} = p_i$ and $f_{ji} = -p_i - k$, respectively. If $k = 0$ we have a tight no-wait constraint (see figure 1).

Let us now consider the blocking constraint, which is used to model the absence of storage capacity between machines. In the rest of this paper, we will distinguish two types of operations, namely *ideal* and *blocking*. We say that an ideal operation o_j remains on machine $M(j)$ from its starting time t_j to its completion time $t_j + p_j$ and then leaves $M(j)$, which becomes immediately available for processing other operations. On the contrary, a blocking operation o_i may remain on machine $M(i)$ even after its completion time, thus blocking it. More precisely, in our model a blocking operation has the following two characteristics:

1. Each blocking operation o_i is associated to another operation, hereafter called $o_{\sigma(i)}$, to be executed on a different machine. In a typical job shop problem, $o_{\sigma(i)}$ is the operation which follows o_i in its job.

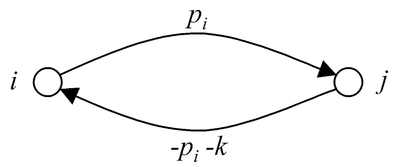

Figure 1. Perishability constraint $t_i + p_i \leqslant t_j \leqslant t_i + p_i + k$.

2. o_i, having completed processing on machine $M(i)$, remains on it until the machine $M(\sigma(i))$ becomes available for processing $o_{\sigma(i)}$. Machine $M(i)$ remains therefore blocked, and cannot process any other operation, until o_i leaves $M(i)$ at time $t_{\sigma(i)}$.

The alternative graph model of this constraint is as follows. Consider two operations o_i and o_j, where o_i is blocking, and $M(i) = M(j)$.

Since o_i and o_j cannot be executed at the same time, we associate with them a pair of alternative arcs. Each arc represents the fact that one operation must be processed before the other. If o_i is processed before o_j, and since o_i is blocking, $M(i)$ can start processing o_j only after the starting time of $o_{\sigma(i)}$, when o_i leaves $M(i)$. Hence, we represent this situation with the alternative arc $(\sigma(i), j)$ having length $a_{\sigma(i)j} = 0$. The other alternative arc is different if o_j is a blocking operation on not. If o_j is blocking, then we add the alternative arc $(\sigma(j), i)$ of length $a_{\sigma(j)i} = 0$. If o_j is ideal, then we add the arc (j, i), whose length is $a_{ji} = p_j$. In figure 2(a) and 2(b) the case of o_j blocking and o_j ideal, respectively, is shown. Here the fixed [alternative] arcs are depicted with solid [dashed] lines.

In figure 3 the alternative graph formulation for a blocking flow shop with three jobs and four machines (and 14 operations, $o_0, o_1, \ldots, o_{13}$) is depicted. In this case, $M(1) = M(5) = M(9)$, $M(2) = M(6) = M(10)$, $M(3) = M(7) = M(11)$, and $M(4) = M(8) = M(12)$. Note that a job, having completed processing on the last machine, leaves the system. Hence, the last machine and operations o_4, o_8, o_{12} are not blocking. For each blocking operation $o_{\sigma(i)} = o_{i+1}$. For each pair of operations to be executed on the same machine there is a pair of alternative arcs. For example, for o_1 and o_5 there is the pair ((2, 5), (6, 1)), for o_4 and o_8 there is the pair ((8, 4), (4, 8)), and so

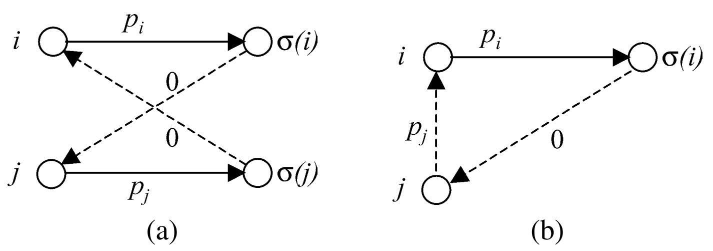

Figure 2. The alternative arcs for a pair of operations o_i and o_j.

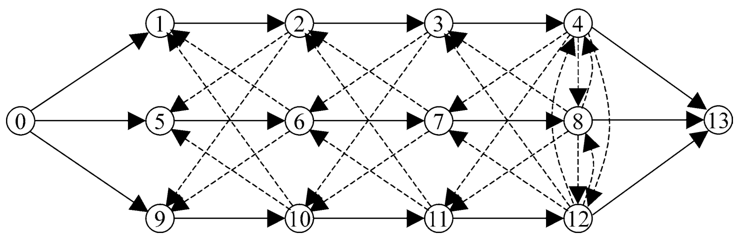

Figure 3. The alternative graph for a blocking flow shop with three jobs and four machines.

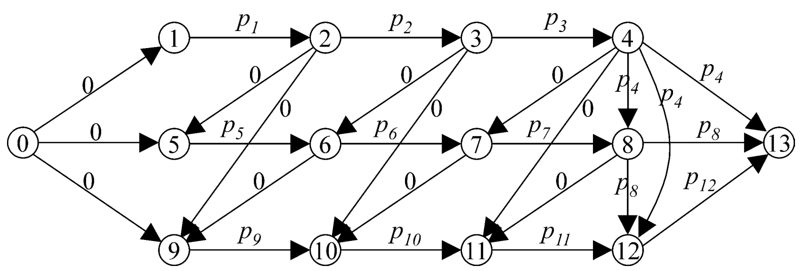

Figure 4. A feasible solution for a blocking flow shop with three jobs and four machines.

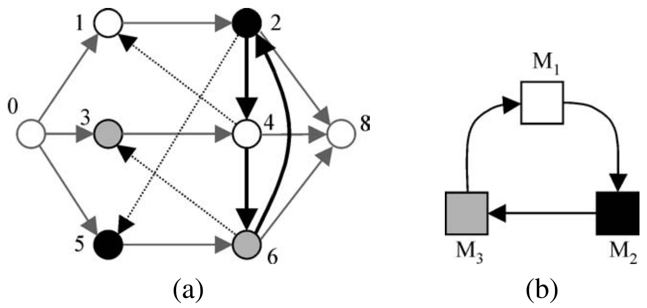

Figure 5. The alternative graph for a cycle of three blocking operations.

on. In total there are 12 pairs of alternative arcs. In figure 4 a feasible solution is shown for this problem.

In this paper we distinguish two cases of blocking: blocking with swap allowed and blocking no swap. A swap is needed whenever there is a cycle of two or more jobs, each one waiting for a machine which is blocked by another job in the cycle. Figure 5(a) shows the alternative graph for a problem with three jobs and eight operations to be processed on the three machines of figure 5(b). Each job consists of two operations, the first job visits M_1 and then M_2, the second M_3 and then M_1, the third job visits M_2 and then M_3. The alternative graph model in this case includes three pairs of alternative arcs: $((2, 4), (4, 1))$, $((4, 6), (6, 3))$ and $((6, 2), (2, 5))$. By selecting the three alternative arcs $(2, 4)$, $(4, 6)$ and $(6, 2)$, we obtain the cycle of figure 5(a). In such a situation all jobs in the cycle must move simultaneously to the next machine, and we call it a *swap*. It is intuitive that, depending on the particular context, a swap may be allowed or not.

The swap is allowed when the jobs can move independently of each other. On the contrary, the swap is not allowed when the jobs can move strictly after that the subsequent resource is become available (e.g., if all movements are performed by a single transportation unit). In the latter case, we say that the operation is a no swap blocking operation. A cycle of blocking operations, in which at least one operation is no swap, is infeasible. When modeling a blocking problem, we must therefore represent two different situations. Since in our formulation a solution is feasible when there are no positive length cycles, a zero length cycle is feasible. This is the case of cycles of blocking operations with swap allowed. In the case of no swap blocking operations, we put a small positive weight ϵ on the corresponding alternative arcs, to model the fact that a job can move strictly after that the subsequent resource is become available. So doing, we make infeasible a solution containing cycles of alternative arcs.

The alternative graph allows to formulate situations by far more general. We refer to (White and Rogers, 1990) and to (Pacciarelli, 2002) for more general examples.

4. Scoring functions

In this section we deal with possible scoring functions based on heuristic procedures. In particular, we discuss three greedy subheuristics for finding a feasible schedule, given a general alternative graph and a k-state. The scoring function is the objective cost returned by the pilot heuristic.

In this section we consider four sub-cases of problem 3.1, namely the ideal job shop problem (IJSP), the blocking no-swap problem (BNSP), the blocking problem with swap allowed (BWSP) and the no-wait job shop problem (NWP). IJSP is the traditional academic job shop problem, in which there are infinite capacity buffers. BNSP and BWSP arise when dealing with a job shop problem in which all the machines are blocking, i.e. without buffers. Finally, NWP arises when dealing with a job shop problem in which all the operations of each job have to be processed without any interruption, i.e. all jobs are no-wait.

An important difference between the IJSP and the other three cases is the following. In the IJSP every feasible state S_k always admits an extension. In the BWSP this is not true in general, even though in (Mascis and Pacciarelli, 2002) a special class of feasible states is shown, called *positional selections*, that always admit an extension. In the BNSP even for this special class of states finding an extension of a k-state is an *NP*-complete problem (Mascis and Pacciarelli, 2002). Finally, for the NWP, finding an extension of a k-state is an *NP*-complete problem even for a problem with three machines and all operations having unit processing times (see Mascis and Pacciarelli, 2002).

From the above discussion it follows that, for the most constrained job shop scheduling problems, it may happen that a constructive algorithm terminates with no feasible solutions. This is the case for our greedy heuristics, and also for the rollout procedure. Note that, for a constructive algorithm, the issues of finding a good quality solution and a feasible solution are conflicting. In fact, the quality issue aims at building partial compact schedules, that easily admit no extension. Hence, when dealing with the most constrained job shop scheduling problems, it is useful to balance the need for compactness of a partial solution and the probability of finding an extension. For this reason we tested three greedy heuristics with different behaviours in terms of compactness of the resulting schedule. In what follows we describe our greedy algorithms and show their respective performance.

The three algorithms are referred to in the following as *Avoid Maximum Current Completion time* (AMCC), *Select Most Critical Pair* (SMCP), and *Select Max Sum Pair* (SMSP), respectively. All the heuristics are based on the idea of repeatedly extending a feasible k-state, given a general instance of problem 3.1. In particular the heuristics

```
Algorithm H($N$, $F$, $A$)

Input: Alternative graph $G = (N, F, A)$.

begin
while $A \neq \emptyset$ do
  begin
  select a pair $((h, k), (i, j)) \in A$
  $F := F \cup \{(i, j)\}$
  $A := A - \{((h, k), (i, j))\}$
  update the values $l(0, u)$ and $l(u, n)$ for each node $u \in N$
  if $l(j, i) + a_{ij} > 0$
    then STOP, the procedure failed in finding a feasible solution
  end
end.
```

Figure 6. The general sketch of the three heuristics.

select one arc at a time, adding it to the set of the fixed arcs. The algorithms iterate until a complete extension is built or a positive length cycle is detected.

We next describe, in figure 6, the general sketch of the three heuristics. Given an alternative graph $G = (N, F, A)$, we define $l(i, j)$ as the length of the longest path in the graph (N, F) from node i to node j. We use the convention that if there is no path from i to j, then $l(i, j) = -\infty$.

In particular, the selection of the unselected pair $((h, k), (i, j)) \in A$ is performed differently in the three heuristics. AMCC selects the pair such that

$$l(0, h) + a_{hk} + l(k, n) = \max_{(u,v)\in A} \big\{ l(0, u) + a_{uv} + l(u, n) \big\}. \tag{1}$$

In other words, given the current c-state S_c, (h, k) is an arc that, if selected, would increase most the length of the longest path in $G(S_c)$. Hence, AMCC selects its alternative (i, j). When choosing a pair $((h, k), (i, j)) \in A$, a tie may occur among the pairs fulfilling (1). In these cases AMCC chooses the one such that $l(0, i) + a_{ij} + l(j, n)$ is minimum. This first heuristic focuses on the quality issue only, by keeping a schedule as compact as possible.

SMCP selects a pair maximizing the quantity:

$$\min\big\{ l(0, h) + a_{hk} + l(k, n), l(0, i) + a_{ij} + l(j, n) \big\}.$$

The selected arc (i, j) is such that $l(0, h) + a_{hk} + l(k, n) \geqslant l(0, i) + a_{ij} + l(j, n)$. This second heuristic focuses mainly on the feasibility issue. It aims at disrupting the current partial schedule as little as possible, by choosing every time the unselected pair that, in the worst case, would increase least the length of the longest path in $G(S_c)$.

The third heuristic, SMSP, selects the pair maximizing the quantity:

$$\big| l(0, h) + a_{hk} + l(k, n) + l(0, i) + a_{ij} + l(j, n) \big|.$$

The selected arc (i, j) is such that $l(0, h) + a_{hk} + l(k, n) \geqslant l(0, i) + a_{ij} + l(j, n)$. SMSP tries to take into account at the same time quality and feasibility issues. It selects every time the unselected pair with the greatest potential effect on the length of the longest path in $G(S_c)$. Hence we expect an intermediate behaviour of this heuristic with respect to the others.

We next discuss the time complexity of the three heuristics. All of them require $O(|A|)$ iterations. In each iteration all the unselected arcs are evaluated, thus requiring time $O(|A|)$, one arc is added to F and all the path lengths $l(0, u)$ and $l(u, n)$ are updated, for each node $u \in N$. Since we are adding a single arc (i, j) to F, the updating phase requires one forward visit in $G(F)$ from j on, to update $l(0, u)$ for all $u \in N$, and one backward visit in $G(F)$ from i back, to update $l(u, n)$ for all $u \in N$. Hence, the updating phase requires $O(|F| + |A|)$ time units. The feasibility check $l(j, i) + a_{ij} > 0$ is performed in constant time during the updating phase. In fact, $l(j, i) + a_{ij} > 0$ if and only if $l(0, i)$ is updated in the forward visit or, equivalently, $l(j, n)$ is updated in the backward visit. In conclusion, the overall complexity of each greedy algorithm is $O(|A||F| + |A|^2)$.

We tested the performance of the heuristics on a sample of instances from the literature on IJSP. In particular, we chose all the instances with 10 jobs and 10 machines for which the optimal solution is known (Mascis and Pacciarelli, 2002). In terms of our parameters $|N|$, $|F|$, $|A|$, all these instances are characterized by $|N| = 102$, $|F| = 110$, and $|A| = 450$. Problems Abz5–6 were generated by Adams, Balas & Zawack (1988), problems Orb01–10 were generated by Applegate and Cook (1991), problems La16–20 are all the instances with 10 jobs and 10 machines from the 40 problems generated by Lawrence (1984). Finally Ft10 is an instance proposed by Fisher and Thompson in a book edited by Muth and Thompson (). This is probably the most well-known instance in the literature on job shop scheduling since it remained unsolved for more than 20 years, until it was finally settled by Carlier and Pinson (1989).

In table 1 we report our results for the three heuristics. In columns 1 and 2 we report the name of the instances and the values of the optimal solutions, respectively. In the subsequent six columns we report, for each instance, the solutions found by the three heuristics and the relative distance from the optimum (percentage).

As expected, from our computational experience AMCC turns out to be the most effective among the three heuristics for IJSP, while SMCP produces the schedules with the largest completion times. However, as already reported by several authors (see, for example (Pinson, 1997)) as a common behaviour of all greedy heuristics, the performance of the three algorithms may be quite erratic. For example, the relative error of AMCC over the 18 instances spans from 0.48% to 15.62%, even if all the instances have the same number of jobs and machines. More important, when dealing with more constrained problems, from BWSP to NWP, AMCC fails very easily in finding feasible solutions. Hence such simple approach does not produce acceptable results as a general solution method when dealing with general alternative graphs. In the next section we adopt these three heuristics as pilot heuristics in our metaheuristic, and report on the performance of the algorithms for a larger set of test cases.

Table 1
Performance of the three heuristics.

Instance	Optimal value	AMCC		SMSP		SMCP	
		Solution	Error (%)	Solution	Error (%)	Solution	Error (%)
Abz5	1234	1346	9.08	1402	13.61	1541	24.88
Abz6	943	985	4.45	978	3.71	1253	32.87
Ft10	930	985	5.91	1013	8.92	1369	47.20
Orb01	1059	1213	14.54	1264	19.36	1444	36.36
Orb02	888	924	4.05	983	10.70	1119	26.01
Orb03	1005	1159	15.32	1244	23.78	1343	33.63
Orb04	1005	1108	10.25	1135	12.94	1303	29.65
Orb05	887	924	4.17	1049	18.26	1120	26.27
Orb06	1010	1126	11.49	1101	9.01	1345	33.17
Orb07	397	459	15.62	470	18.39	475	19.65
Orb08	899	981	9.12	1072	19.24	1281	42.49
Orb09	934	1015	8.67	1027	9.96	1238	32.55
Orb10	944	1055	11.76	1152	22.03	1076	13.98
La16	945	979	3.60	1060	12.17	1144	21.06
La17	784	800	2.04	892	13.78	1041	32.78
La18	848	916	8.02	976	15.09	1127	32.90
La19	842	846	0.48	932	10.69	991	17.70
La20	902	930	3.10	964	6.87	1053	16.74
Average error			7.87		13.81		28.88
Maximum error			15.62		23.78		47.20

5. Rollout scheme

In this section we describe our rollout scheme for finding good feasible solutions, given a general alternative graph. A discussion on the computational experience follows. As in the previous section, we refer in particular to four cases of the job shop problem, namely IJSP, BWSP, BNSP, and to NWP. We report on the performance of our rollout metaheuristic on the 18 instances of IJSP from section 4, plus other 18 instances for each one of the other three problems. The test instances for blocking and no-wait job shop problems are obtained as follows. From each instance of IJSP we derived four instances, each obtained by considering all machines as ideal, blocking with swap allowed, blocking no swap, and no-wait, respectively. To the best of our knowledge these are the only non-trivial instances of blocking and no-wait job shop problems for which the optimal solution is known (Mascis and Pacciarelli, 2000). This is particularly useful in order to analyse the performance of the method.

As already observed in section 2, a rollout algorithm is characterized by the scoring function. We considered three different scoring functions $\mathrm{H}(N, F, A)$ by running a single heuristic at a time, among those of section 4. The rollout algorithm is represented in figure 7. We indicate with $\hat{y}$ the alternative arc of y, i.e. $(y, \hat{y}) \in A$.

Besides the basic rollout scheme of figure 7, we implemented some additional simple functions in order to speed up the algorithm. For example, the execution of

```
Algorithm Rollout
Input: Alternative graph $G = (N, F, A)$.
begin
while $A \neq \emptyset$ do
   begin
   $best = +\infty$
   forall $y \in A$ do
      begin
      if $H(N, F \cup \{y\}, A - (y, \hat{y})) < best$ then
         begin
         $y_{best} = y$
         $best = H(N, F \cup \{y\}, A)$
         end
      end
   $F = F \cup \{y_{best}\}$
   $A = A - (y_{best}, \hat{y}_{best})$
   end
end
```

Figure 7. The basic rollout scheme.

H(N, F, A) is halted whenever the cost of the current partial solution exceeds the current upper bound of the problem (i.e., the best value obtained so far by the rollout algorithm). Instead of evaluating the scoring function for all $y \in A$, we limit the evaluation to 20% of all arcs, randomly chosen. More in details, at each step of the algorithm, the total number of alternative arcs is $2|A|$, $|A|$ of which are selected in the current best solution. We look for a solution, better than the current best one, by selecting one arc of the $|A|$ arcs that are forbidden in the current best solution. In other words, at each step of the algorithm we explore only the set of solutions that are certainly different from the current best one. If we do not succeed in finding a better solution, then we keep the current best one. For each of the $|A|$ remaining k-states we generate a random number in [0, 1] and compute the scoring function only if the number is smaller or equal to 0.4. We experienced that this choice speeds up the algorithm without affecting significantly the overall performance. In fact, when computing the scoring function for all the arcs in each iteration, we observed that the current best solution was found several times, starting from different k-states. We therefore filtered the set of candidate arcs to speed up the method. A filter is also used in (Duin and Voß, 1999) to speed up the pilot method, and in (Ow and Morton, 1988) to speed up a beam search procedure for solving scheduling problems.

We also used a simple constraint propagation rule from (Carlier and Pinson, 1989, 1994). Given an alternative graph, a k-state S_k, an upper bound B (the cost of the best solution found so far), and a pair of unselected alternative arcs $((i, j), (h, k)) \in A$, we

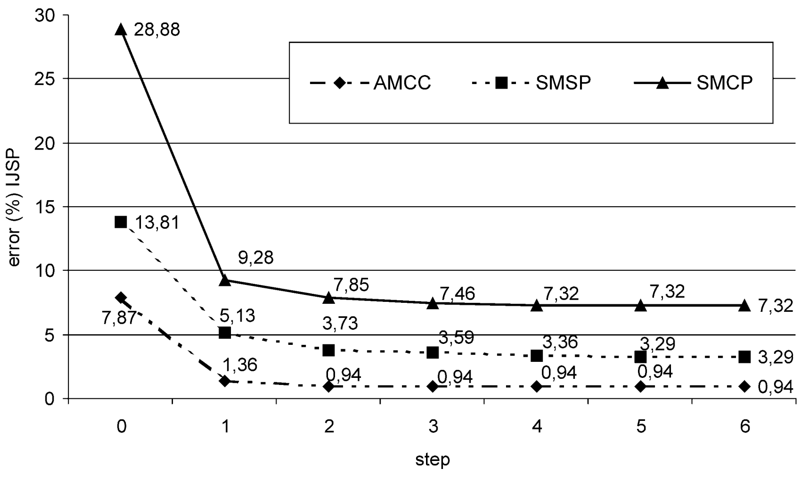

Figure 8. Performance of the three heuristics in the rollout scheme for IJSP.

say that the arc (i, j) is implied by S_k if

$$l(0, h) + a_{hk} + l(k, n) \geqslant B.$$

In other words, starting from S_k, no solution with cost smaller than B exists if arc (h, k) is selected, and therefore it is necessary to select arc (i, j). Since the rollout algorithm may require a considerable computing time at each iteration, we introduced two stopping criteria. First, as already observed in section 4, in the most constrained cases it may occur that no feasible solution is found after one or more iterations. Hence, when the heuristic fails in finding a feasible solution after a rollout iteration, we fix an arc arbitrarily and iterate. After eight rollout iterations, if no solution has been still found, then the algorithm is stopped. When at least one feasible solution is available, the rollout algorithm is stopped after two non-improving iterations.

All our experiments were executed on a PC equipped with a Pentium II 350 MHz processor, and 128 MB of RAM. Figures 8–11 report on our computational experience. We next briefly comment our results. Figure 8 shows the average error, with respect to the optimal solutions, of the three rollout schemes for the 18 instances of IJSP from section 4. At iteration 0 the average error of the stand-alone heuristics is reported. At iterations 1–6, figure 8 shows the improvement of the average error for the best solution found by the rollout algorithm when using each one of the three scoring functions. We observe that the rollout scheme is an effective way for improving the value found by each heuristic. Moreover, the improvement is quite effective, even after a single iteration. In this case, by using the scoring function AMCC, the relative error is smaller than 1% already at the second iteration of the rollout scheme. On the other hand, the rollout scheme performs differently with the scoring function being used. A good heuristic (e.g., AMCC) provide better overall performance for the rollout scheme. When the scoring function provides poor performance, such as for SMCP, then also the rollout scheme

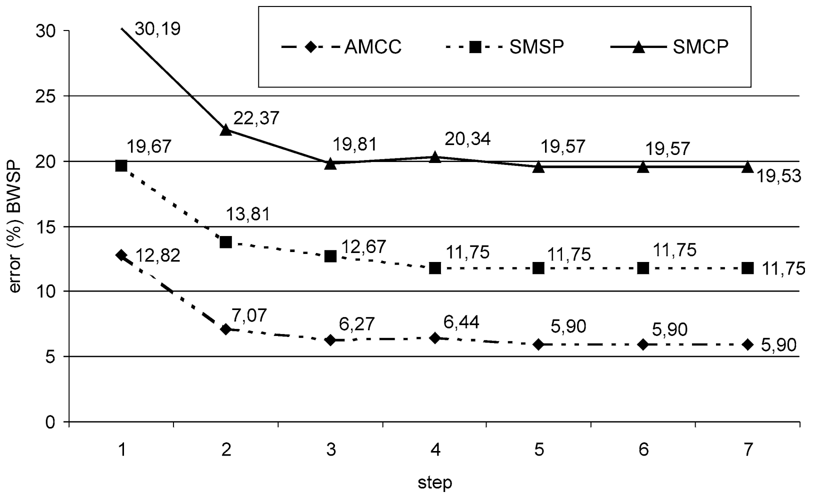

Figure 9. Performance of the three heuristics in the rollout scheme for BWSP.

provides poor overall performance, even if it is still able to improve the performance of the stand-alone heuristic.

Figure 9 shows the performance of the three rollout schemes for the 18 instances of BWSP. We recall that these instances are obtained from the 18 instances of IJSP from section 4, by considering all the machines as blocking with swap allowed. Note that, due to the different combinatorial structure of this problem, in this case the greedy heuristics can terminate with no feasible solutions. In fact, all the three heuristics fail very easily in finding feasible solutions. In this case the rollout scheme is very effective in improving the number of feasible solutions. After a single iteration a feasible solution is found in almost all cases, and at the fourth iteration a feasible solution is found for all instances and for all the rollout schemes. Also in this case the best performance is provided by the best heuristic, with an average error smaller than 5%. Note that, in this case we report in figure 9 the average error of the algorithm, only for those instances for which a feasible solution is found. Hence, it may happen that the average error increases from a iteration to the next when the number of feasible solutions increases, as for AMCC and SMCP at the fourth iteration of the rollout scheme.

Figure 10 shows the performance of the three rollout schemes for the 18 instances of BNSP. Also these instances are obtained from the 18 instances of IJSP from section 4, by considering all the machines as blocking no swap. In this case, all the three heuristics fail in finding feasible solutions at iteration 0. In this case the most conservative heuristics, SMSP and SMCP, exhibit a better performance in terms of feasible solutions. From the point of view of the average error, in this case AMCC and SMSP are comparable, even though the average computed for SMSP is taken over a larger set of feasible instances than for AMCC.

Figure 11 shows the performance of the three rollout schemes for the 18 instances of NWP. Also these instances are obtained from the 18 instances of IJSP from section 4,

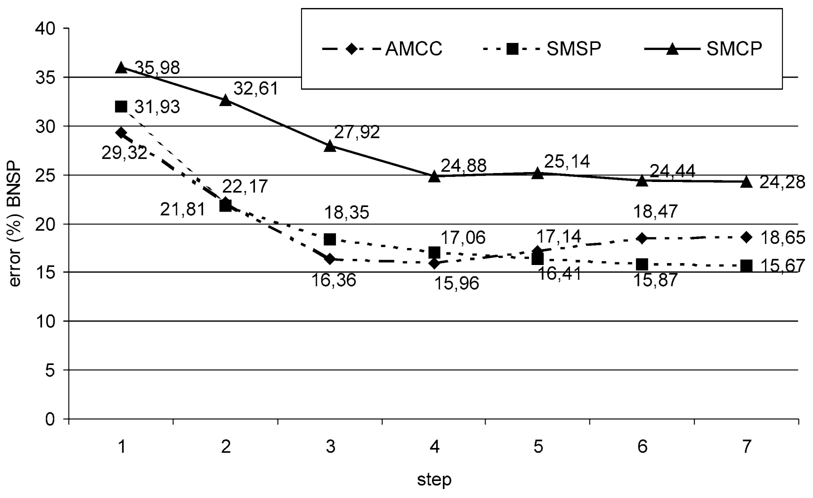

Figure 10. Performance of the three heuristics in the rollout scheme for BNSP.

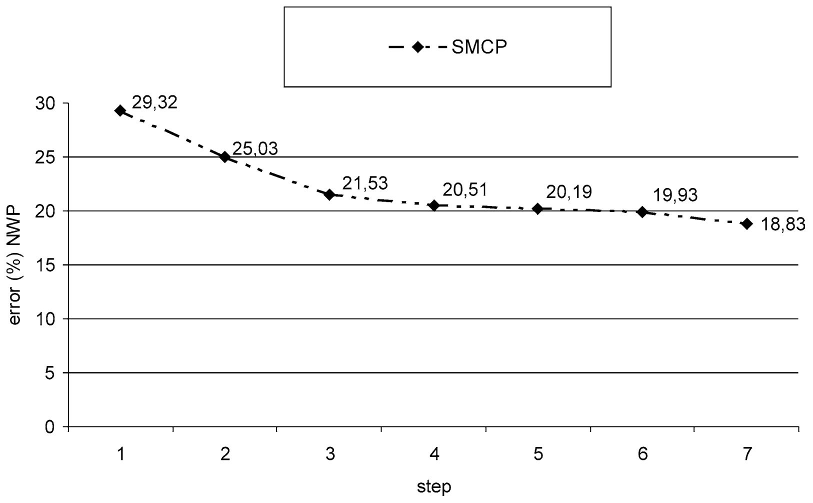

Figure 11. Performance of SMCP in the rollout scheme for NWP.

by considering all the jobs as no-wait. For this very constrained problem, only the most conservative heuristic, SMCP, is able to find a good number of feasible solutions. In fact a feasible solution is found for all instances within the first two iterations of the rollout. The other two heuristics are unable to find a significant number of solutions after eight iterations of rollout. In figure 11 the average performance of SMCP is shown.

In table 2 we summarize the results of the three rollout schemes for the four problems considered. In particular, in column 1 the name of the scoring function used by the rollout scheme is reported, from columns 2–5 we report the average error obtained over the 18 instances of IJSP, BWSP, BNSP, and NWP, respectively. From columns 6–9

Table 2
Performance of the three heuristics on the four problems.

Heuristic	Average error (%)				Maximum error (%)			
	IJSP	BWSP	BNSP	NWP	IJSP	BWSP	BNSP	NWP
AMCC	0.94	5.90	18.65	N.A.	2.59	9.53	39.07	N.A.
SMSP	3.29	11.75	15.67	N.A.	8.22	21.28	30.88	N.A.
SMCP	7.32	19.53	24.28	18.83	12.82	27.08	45.26	38.40

we report the maximum error obtained over the 18 instances of IJSP, BWSP, BNSP, and NWP, respectively. From these results we can conclude that the rollout scheme using AMCC is very effective in solving the easiest problems, IJSP and BWSP. In particular for IJSP, the error is smaller than 1% in the average case, smaller than 3% in the worst case. Hence, the rollout method results in a robust behaviour over the different instances, comparable with those of the best solution approaches from the literature on job shop scheduling (see, e.g., (Vaessens, Aarts, and Lenstra, 1992)). When the problems become more constrained, as for BNSP and NWP, then more conservative heuristics provide the best results. In this case the solution algorithm has to deal with the conflicting issues of feasibility and quality of the solutions, and therefore the best solutions found are quite far from the optimum. However, in these cases we cannot compare our results with those of other heuristic solution approaches, since we found no published results for BWSP, BNSP, and NWP in the literature.

From tables 3–6, we report the beahaviour of our rollout algorithm for the 72 instances, when using the best performing subheuristic. In particular, the name of each instance and the optimal value are given in columns 1 and 2, respectively. The subsequent columns report the best solution found at the end of each iteration, and the computation time of each iteration in seconds. Iteration 0 coincides with the stand-alone heuristic. We report a time 0 when the rollout did not perform a step due to one of the stopping criteria.

6. Conclusions

In this paper we studied the applicability of rollout metaheuristics to solve a general class of scheduling problems, represented by the alternative graph formulation. Important particular cases are the job shop scheduling problem with unlimited capacity buffers (IJSP), and the job shop problems with blocking and no-wait constraints (BWSP, BNSP, NWP). An important difference between the former and the latter cases is that, while in IJSP every feasible state S_k always admits an extension, in the other cases this is not true in general. Hence, in such cases one has to deal with the conflicting issues of feasibility and compactness of the solutions.

From our computational experience several general conclusions can be drawn. Strong points of rollout algorithms include ease of implementation and broad applicability. Moreover, by using general models and heuristic procedures, as in the case of the

Table 3
Results for IJSP instances, obtained with AMCC subheuristic.

Instance	Optimal value	Iter. 0		Iter. 1		Iter. 2		Iter. 3		Iter. 4		Iter. 5	
		best	Time	best	Time	best	Time	best	Time	best	Time	best	Time
Abz5	1234	1346	<1	1250	19	1250	19	1250	19	1250	0	1250	0
Abz6	943	985	<1	947	18	947	17	947	17	947	0	947	0
Ft10	930	985	<1	943	23	943	22	943	22	943	0	943	0
Orb01	1059	1213	<1	1094	23	1081	22	1081	22	1081	22	1081	0
Orb02	888	924	<1	897	17	889	19	889	17	889	17	889	0
Orb03	1005	1159	<1	1047	23	1031	23	1031	22	1031	21	1031	0
Orb04	1005	1108	<1	1033	21	1006	20	1006	20	1006	19	1006	0
Orb05	887	924	<1	890	20	890	19	890	19	890	0	890	0
Orb06	1010	1126	<1	1032	22	1026	21	1026	22	1026	22	1026	0
Orb07	397	459	<1	403	20	403	19	403	18	403	0	403	0
Orb08	899	981	<1	919	21	919	22	919	22	919	0	919	0
Orb09	934	1015	<1	945	21	943	22	943	21	943	22	943	0
Orb10	944	1055	<1	944	19	944	19	944	19	944	0	944	0
La16	945	979	<1	956	20	956	18	956	18	956	0	956	0
La17	784	800	<1	788	18	788	19	788	18	788	0	788	0
La18	848	916	<1	852	18	848	18	848	17	848	16	848	0
La19	842	846	<1	842	18	842	18	842	18	842	0	842	0
La20	902	930	1	907	19	907	18	907	17	907	0	907	0

Table 4
Results for BWSP instances, obtained with AMCC subheuristic.

Instance	Optimal value	Iter. 0		Iter. 1		Iter. 2		Iter. 3		Iter. 4		Iter. 5	
		best	Time	best	Time	best	Time	best	Time	best	Time	best	Time
Abz5	1468	∞	<1	1636	24	1595	24	1595	19	1595	18	1595	0
Abz6	1145	∞	<1	1422	25	1222	23	1222	20	1222	20	1222	0
Ft10	1068	∞	1	1225	22	1145	25	1144	26	1144	25	1144	25
Orb01	1175	∞	1	1387	19	1287	25	1287	25	1287	25	1287	0
Orb02	1041	∞	<1	1142	22	1110	23	1110	19	1110	18	1110	0
Orb03	1160	∞	<1	1314	12	1202	24	1192	22	1192	22	1188	21
Orb04	1146	∞	<1	∞	24	1313	22	1223	24	1223	22	1223	21
Orb05	995	∞	<1	1098	22	1092	18	1083	12	1083	<1	1083	<1
Orb06	1199	∞	<1	1477	21	1317	26	1317	26	1278	25	1278	23
Orb07	483	∞	<1	550	25	517	24	517	21	517	20	517	0
Orb08	995	∞	<1	∞	15	∞	20	∞	0	∞	0	∞	0
Orb09	1039	∞	<1	1046	22	1046	22	1046	21	1046	0	1046	0
Orb10	1146	∞	<1	1153	20	1153	22	1153	21	1153	0	1153	0
La16	1060	∞	<1	1189	26	1109	25	1109	21	1109	20	1109	0
La17	929	∞	<1	1071	24	993	22	982	21	982	21	982	21
La18	1025	∞	<1	1154	24	1114	23	1114	22	1114	21	1114	0
La19	1043	∞	<1	1199	22	1144	25	1115	20	1115	18	1115	18
La20	1060	∞	1	1166	19	1118	23	1118	21	1118	20	1118	0

Table 5
Results for BNSP instances, obtained with SMSP subheuristic.

Instance	Optimal value	Iter. 0		Iter. 1		Iter. 2		Iter. 3		Iter. 4		Iter. 5	
		best	Time	best	Time	best	Time	best	Time	best	Time	best	Time
Abz5	1641	∞	<1	1949	11	1874	20	1838	19	1838	19	1838	18
Abz6	1249	∞	<1	1718	15	1419	20	1419	19	1419	18	1419	0
Ft10	1158	∞	<1	∞	11	1629	12	1447	18	1447	18	1447	17
Orb01	1256	∞	<1	2246	4	1572	13	1435	16	1342	11	1335	11
Orb02	1144	∞	<1	1450	12	1422	18	1402	14	1370	17	1370	15
Orb03	1311	∞	<1	∞	3	∞	4	∞	5	∞	4	∞	5
Orb04	1246	∞	<1	1658	13	1599	21	1511	19	1460	17	1460	16
Orb05	1203	∞	<1	1623	11	1414	14	1414	14	1414	12	1414	0
Orb06	1266	∞	<1	1675	12	1462	17	1462	17	1448	17	1448	17
Orb07	527	∞	<1	568	12	565	17	565	15	565	15	565	0
Orb08	1139	∞	1	∞	7	∞	6	∞	10	∞	10	1223	9
Orb09	1130	∞	<1	1477	7	1392	18	1329	17	1329	18	1329	19
Orb10	1367	∞	<1	∞	4	1789	6	1789	19	1789	13	1789	0
La16	1148	∞	<1	1542	12	1313	21	1272	20	1231	17	1231	16
La17	968	∞	<1	1266	14	1199	19	1150	12	1146	16	1146	12
La18	1077	∞	<1	1406	11	1344	17	1334	18	1334	15	1334	7
La19	1102	∞	<1	1314	9	1314	21	1314	20	1314	0	1314	0
La20	1118	∞	<1	1468	18	1408	21	1379	18	1366	18	1357	18

Table 6
Results for NWP instances, obtained with SMCP subheuristic.

Instance	Optimal value	Iter. 0		Iter. 1		Iter. 2		Iter. 3		Iter. 4		Iter. 5	
		best	Time	best	Time	best	Time	best	Time	best	Time	best	Time
Abz5	2150	∞	<1	∞	7	3040	9	2726	11	2726	11	2726	10
Abz6	1718	∞	<1	2207	3	2178	11	2178	9	2178	10	2178	0
Ft10	1607	∞	<1	∞	6	2190	3	1882	10	1882	4	1882	5
Orb01	1615	∞	<1	2256	6	1708	8	1708	9	1708	9	1708	0
Orb02	1485	∞	<1	1814	9	1814	9	1814	8	1814	0	1814	0
Orb03	1599	∞	<1	1849	14	1768	11	1768	12	1768	16	1768	0
Orb04	1653	∞	<1	2057	6	1836	9	1836	8	1836	6	1836	0
Orb05	1365	∞	<1	∞	5	1609	6	1609	5	1609	4	1609	0
Orb06	1555	∞	<1	2003	5	1793	10	1793	5	1793	1	1793	0
Orb07	689	∞	<1	880	5	880	13	880	10	880	0	880	0
Orb08	1319	∞	<1	1694	6	1694	10	1542	5	1542	7	1542	11
Orb09	1445	∞	<1	2189	7	2051	12	2029	4	2029	3	1945	11
Orb10	1557	∞	<1	1921	8	1912	11	1773	7	1773	8	1773	10
La16	1575	∞	<1	1659	5	1659	11	1659	8	1659	0	1659	0
La17	1371	∞	<1	2040	5	1665	11	1665	9	1665	9	1665	0
La18	1417	∞	<1	∞	4	1981	4	1963	13	1950	12	1950	10
La19	1482	∞	<1	2028	5	2020	12	1950	12	1690	11	1690	8
La20	1526	∞	1	∞	6	2130	7	2112	12	2112	8	2112	6

alternative graph formulation, it is possible to develop a unique solution scheme for a broad class of problems. On the other hand, the method performs differently with the heuristics being used. When the heuristics are always able to find a feasible solution, as in IJSP, then better heuristics are more effective in guiding the rollout algorithm towards better solutions. However, the rollout scheme is by far more robust than the single stand-alone heuristics, both in terms of average quality of the solutions found and worst case performance. When dealing with more and more constrained problems, such as BWSP, BNSP, and NWP, then the feasibility issue becomes more and more important. In such cases the best overall performance is obtained by using more conservative heuristics, able to find a larger number of feasible solutions. Also in this case, however, the rollout scheme allows to find a larger number of feasible solutions and to reduce the gap with the optimum, with respect to the single stand-alone heuristics.

A comparison with other metaheuristic approaches is possible only for IJSP, since we found no comparable results for the more constrained versions of the job shop problem. To this aim, however, we observe that our rollout algorithm was developed to deal with general alternative graphs. Hence, it does not take fully advantage of the particular structure of the IJSP, thus resulting in less efficient data structures, and significantly longer computing times. From the viewpoint of the solution quality, the rollout scheme outperforms many other solution approaches from the literature on job shop scheduling, and its results are not far from those of the best existing algorithms. However, further research is needed in order to speed up the procedure. Different possibilities include more sophisticated constraint propagation rules and filters, or faster heuristics.

Acknowledgments

The authors gratefully acknowledge the helpful comments and constructive suggestions of the associate editors and the anonymous referees.

References

Adams, J., E. Balas, and D. Zawack. (1988). "The Shifting Bottleneck Procedure for Job Shop Scheduling." *Management Science* 34(3), 391–401.

Applegate, D. and W. Cook. (1991). "A Computational Study of the Job Shop Scheduling Problem." *ORSA Journal on Computing* 3(2), 149–156.

Balas, E. (1979). "Disjunctive Programming." *Annals of Discrete Mathematics* 5, 3–51.

Balas, E. and A. Vazacopoulos. (1998). "Guided Local Search with Shifting Bottleneck for Job Shop Scheduling." *Management Science* 44(2), 262–275.

Bertsekas, D.P. and J.N. Tsitsiklis. (1996). *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Bertsekas, D.P., J.N. Tsitsiklis, and C. Wu. (1997). "Rollout Algorithms for Combinatorial Optimization." *Journal of Heuristics* 3, 245–262.

Błażewicz, J., W. Domschke, and E. Pesch. (1996). "The Job Shop Scheduling Problem: Conventional and New Solution Techniques." *European Journal of Operational Research* 93(1), 1–33.

Carlier, J. and E. Pinson. (1989). "An Algorithm for Solving the Job-Shop Problem." *Management Science* 35(2), 164–176.

Carlier, J. and E. Pinson. (1994). “Adjustment of Heads and Tails for the Job-Shop Problem.” *European Journal of Operational Research* 78, 146–161.

Cassandras, C.G., S. Lafortune, and G.J. Olsder. (1995). “Introduction to the Modelling, Control and Optimization of Discrete Event Systems.” In A. Isidori (ed.), *Trends in Control*. London: Springer, pp. 217–292.

Dorndorf, U. and E. Pesch. (1995). “Evolution Based Learning in a Job Shop Scheduling Environment.” *Computers & Operations Research* 22, 25–40.

Duin, C. and S. Voß. (1999). “The Pilot Method: A Strategy for Heuristic Repetition with Application to the Steiner Problem in Graphs.” *Networks* 34, 181–191.

Fox, M.S. (1983). “Constraint Directed Search: A Case Study of Job Shop Scheduling.” Ph.D. Thesis, Carnagie Mellon University, Pittsburg, PA.

Glover, F. (1986). “Future Paths for Integer Programming and Links to Artifical Intelligence.” *Computers & Operations Research* 13, 533–549.

Hall, N.J. and C. Sriskandarajah. (1996). “A Survey on Machine Scheduling Problems with Blocking and No-Wait in Process.” *Operations Research* 44(3), 510–525.

Jain, A.S. and S. Meeran. (1999). “Deterministic Job-Shop Scheduling: Past, Present and Future.” *European Journal of Operational Research* 113, 390–434.

Lawrence, S. (1984). Supplement to *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*, GSIA, Carnagie Mellon University, Pittsburg, PA.

Lowerre, B.T. (1976). “The HARPY Speech Recognition System.” Ph.D. Thesis, Carnagie Mellon University, Pittsburg, PA.

Mascis, A. and D. Pacciarelli. (2000). “Machine Scheduling via Alternative Graphs.” Report DIA-46-2000, Dipartimento di Informatica e Automazione, Università Roma Tre.

Mascis, A. and D. Pacciarelli. (2002). “Job Shop Scheduling with Blocking and No-Wait Constraints.” *European Journal of Operational Research* 143(3), 498–517.

Muth, J.F. and G.L. Thompson (eds.). (1963). *Industrial Scheduling*. Amsterdam: Kluver Academic.

Nowicki, E. and C. Smutnicki. (1996). “A Fast Taboo Search Algorithm for the Job Shop Scheduling Problem.” *Management Science* 42(6), 797–813.

Nuijten, W. and C. Le Pape. (1998). “Constraint-Based Job Shop Scheduling with Ilog Scheduler.” *Journal of Heuristics* 3, 271–286.

Ow, P.S. and T.E. Morton. (1988). “Filtered Beam Search in Scheduling.” *International Journal of Production Research* 26, 297–307.

Pacciarelli, D. (2002). “The Alternative Graph Formulation for Solving Complex Factory Scheduling Problems.” *International Journal of Production Research* 40(15), 3641–3653.

Pinson, E. (1997). “The Job Shop Scheduling Problem: A Coincise Survey and Some Recent Developments.” In P. Chrétienne, E.G. Coffman, J.K. Lenstra, and Z. Liu (eds.), *Scheduling Theory and Its Applications*. New York: Wiley, pp. 277–294.

Roy, B. and B. Sussman. (1964). “Les Problèmes D’ordonnancement avec Contraintes Disjonctives.” Note DS No. 9bis, SEMA, Paris.

Sadeh, N., K. Sycara, and Y. Xiong. (1995). “Backtracking Techniques for the Job Shop Scheduling Constraints Satisfaction Problem.” *Artificial Intelligence* 76, 455–480.

Schutten, J.M.J. (1998). “Practical Job Shop Scheduling.” *Annals of Operations Research* 83, 161–177.

Vaessens, R.J.M., E.H.L. Aarts, and J.K. Lenstra. (1992). “Job Shop Scheduling by Local Search.” *INFORMS Journal on Computing* 8(3), 302–317.

Voß, S., S. Martello, I.H. Osman, and C. Roucairol (eds.). (1999). *Meta-Heuristics:Advances and Trends in Local Search Paradigms for Optimization*. Boston: Kluwer Academic.

White, K.P. and R.V. Rogers. (1990). “Job-Shop Scheduling: Limits of the Binary Disjunctive Formulation.” *International Journal of Production Research* 28(12), 2187–2200.