ELSEVIER European Journal of Operational Research 83 (1995) 347-364

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Job-shop scheduling: Computational study of local search and large-step optimization methods

Helena Ramalhinho Lourenço [*]

School of Operations Research and Industrial Engineering, Cornell University, Ithaca NY 14853, USA

Abstract

We present a computational study of different local search and large-step optimization methods to solve the job-shop scheduling problem. We review local optimization methods and propose a two-phase optimization method, known as large-step optimization, which has recently been introduced for the traveling salesman problem. The first phase of this new method consists of a large optimized transition in the current solution, while the second phase is basically a local search method. We present extensive computational results obtained from various combinations of local search and large-step optimization techniques. From the computational results we can conclude that the large-step optimization methods outperform the simulated annealing method and find more frequently an optimal schedule than the other studied methods.

Keywords: Scheduling theory; Heuristics; Computational analysis

1. Introduction

In the job-shop scheduling problem we are given a set of m machines $\{M_1, \dots, M_m\}$ and a set of n jobs $\{J_1, \dots, J_n\}$. Each job J_j, $j = 1, \dots, n$, consists of a sequence of m_j operations $O_{1j}, \dots, O_{m_j j}$, where O_{ij} is an operation of job J_j to be processed on machine μ_{ij} for a given uninterrupted processing time p_{ij}, where $\mu_{ij} \neq \mu_{i+1j}$, for $i = 1, \dots, m_j$, $j = 1, \dots, n$.

The operations of each job must be processed in the given sequence. Each machine M_i, $i = 1, \dots, m$, can process at most one operation at a time, and at most one operation of each job J_j, $j = 1, \dots, n$, can be processed at a time. Let C_{ij} be the completion time of operation O_{ij}. The objective is to get a schedule that minimizes the maximum completion time $C_{\max} = \max_{i,j} C_{ij}$. The optimal value is denoted by $C^*_{\max}$. A schedule is an allocation of a single time interval for each operation. A more useful representation is provided by the disjunctive graph model due to Roy and Sussmann [27]. The disjunctive graph can be represented as follows: $G = (O, A, E)$, where O is the vertex set corresponding to the operation set, A is the arc set corresponding to the job precedence constraints and E is the edge set corresponding to the machine capacity constraints.

In the disjunctive graph, the basic scheduling

[*] Research supported by Junta Nacional de Investigação Científica e Tecnológica, Lisbon, Portugal.
New address: Departamento de Estatística e Investigação Operacional, Faculdade de Ciências, Universidade de Lisboa, Campo Grande, C/2, 1700 Lisbon, Portugal.

decision corresponds to orienting each edge in one direction or the other. A schedule is obtained by orienting all edges. The orientation is feasible if the resulting directed graph is acyclic, and its length is equal to the weight of a maximum weight path from s to t in this acyclic graph.

The job-shop scheduling problem is NP-hard in the strong sense. The reduction was obtained from the 3-partition problem [16]. Furthermore, most of the special cases of this problem are NP-hard or strongly NP-hard.

Several branch-and-bound methods have been developed for solving the job-shop scheduling problem to optimality; the more recent ones were proposed by Carlier and Pinson [9,10], Applegate and Cook [4], and Brucker, Jurish and Sievers [7]. These methods require a large amount of computation time and therefore it is not practical to apply them to many instances of even modest size. Carlier and Pinson [9] solved the famous 10-job 10-machine instance of the job-shop scheduling problem proposed by Fisher and Thompson [15], which remained unsolved for a long period of time and largely motivated the development of algorithms for this scheduling problem. Applegate and Cook presented 7 problems that they could not solve to optimality, where the smallest one has 10 machines and 15 jobs. Recently, Lourenço [22] proposed a new lower bound for the job-shop scheduling problem, which is a strengthening of one frequently used, and is based on solving a one-machine scheduling problem with additional constraints corresponding to minimal lags of time between the process of some pairs of jobs.

Most of the approximation methods proposed for the problem are tailored methods, i.e., methods developed specifically for the problem in consideration, for instance the priority dispatching rules [12] and the shifting bottleneck procedure proposed by Adams, Balas and Zawack [2]. Applegate and Cook [4] presented an approximation method, the shuffle algorithm, which is a partial enumeration heuristic based on the branch-and-bound method by Carlier and Pinson [9] where one or more machine orderings are fixed and the remaining ones are optimally completed using a branch-and-bound method.

Recently, there has been an increasing interest in methods based on local search optimization techniques, in which a given solution is iteratively changed by making local modifications. The most well-known methods are the standard local improvement method and the simulated annealing method.

In practice, the job-shop scheduling problem appears frequently not as formulated before, the standard formulation, but with some complicated constraints, different objective functions or small different variations of the problem, as for example in the presence of set-up times. In this case, local optimization methods are a good option since they are easily implemented and require little insight into the combinatorial structure of the problem. Note that the tailored and the branch-and-bound methods usually are not as easily generalized to incorporate these complications as local optimization methods.

Other techniques have been recently applied to the job-shop scheduling problem, as for example tabu search and genetic algorithms. Tabu search is close to local improvement methods, but the interesting feature is the construction of a list of tabu moves. These are moves which are not allowed during a certain number of iterations. Dell'Amico and Trubian [13] and Barnes [6] present tabu search methods for the job-shop scheduling problem. Dorndorf and Pesch [14] present a probabilistic learning strategy based on genetic algorithms to the same problem. Aarts, van Laarhoven, Lenstra and Ulder [1] compare several local optimization methods with each other by applying them to the job-shop scheduling problem. The methods considered are based on a multi-start local search algorithm, simulated annealing, threshold accepting and genetic algorithms.

The paper is organized as follows: in Section 2 we review the known heuristic methods as the priority dispatching rules [12], the shifting bottleneck procedure [2], local improvement and simulated annealing [28]. In Section 3 we propose a variant of local improvement methods to solve the job-shop scheduling, designated by large-step optimization methods and initially presented by Martin, Otto and Felden [24] and Johnson [17] for the traveling salesman problem. In Section 4, we present an extensive computational study where we applied different combinations of the above methods to well-known instances of the job-shop scheduling problem. Finally in Section 5 we present the conclusions of this study.

2. Heuristic and local search optimization methods

2.1. The priority dispatching rules and the shifting bottleneck procedure

The priority dispatching rules are greedy methods that sequentially select and schedule operations according to some priority function. As examples of priority functions we have considered the most work remaining and the longest processing time. At each iteration, the operation with maximum priority among all operations that can be scheduled is selected and scheduled as soon as possible. The priority dispatching rule is easily implemented in $O(nm \log n)$.

Adams, Balas and Zawack [2] developed a shifting bottleneck heuristic which employs a combination of schedule constraints and iterative improvements guided by solutions to one-machine problems. This approach is strongly tailored to the job-shop scheduling problem. The shifting bottleneck procedure can be described as follows: suppose that we have scheduled all jobs $\{J_1, \dots, J_n\}$ on the machines $M_1, \dots, M_k$ (that is, for each of these machines we have fixed the order in which we process the jobs). Now, for each of the remaining machines, not yet scheduled, we obtain the values of the release dates and delivery times, respecting the schedules on the machines $M_1, \dots, M_k$, and solve to optimality the one-machine scheduling problem associated with each unscheduled machine, using Carlier's algorithm [8]. Then, we order the operations of the machine with the greatest optimal value obtained, designated as the bottleneck machine, and perform a local reoptimization of the k machines scheduled before. The entire process is repeated until all m machines are scheduled. Recently, Balas, Lenstra and Vazacopoulos [5] presented a modified version of the shifting bottleneck procedure; instead of using the simple one-machine scheduling problem, they use the one-machine scheduling problem with lags to identify and schedule the bottleneck machine. Both the shifting bottleneck procedure and the modified version give very good schedules for the job-shop scheduling problem, but are computationally more expensive than the priority dispatching rules, since they are based on a branch-and-bound method.

2.2. Local improvement methods

Local improvement methods are iterative methods where initially a feasible solution of the problem in question is obtained and at each step we make one local modification, or transition, of a prespecified type in the current solution. If an improvement in the cost function is obtained, then we accept the new solution as the current one, and otherwise we reject it. The algorithm terminates when we cannot improve the current solution by performing one transition, i.e., when we have a local optimal solution. Note that a local optimal solution is not necessarily a global optimal one. The structure of a local improvement algorithm is presented in Fig. 1.

First of all, we have to define a set of solutions and a cost function of the problem, which usually

```
1. Get an initial schedule $\sigma$.
2. While there is an untested neighbor of $\sigma$ do the following:
   2.1 Let $\sigma'$ be an untested neighbor of $\sigma$.
   2.2 If $C_{max}(\sigma') < C_{max}(\sigma)$ set $\sigma = \sigma'$.
3. Return $\sigma$.
```

Fig. 1. Local improvement method.

is associated with the objective function for the optimization problem. Next, we will define a fundamental concept in local optimization which is the concept of a neighborhood structure. This is a relation, not necessarily symmetric, which contains the pair of solutions (σ_i, σ_j) if and only if σ_j is obtained from a transition of σ_i.

Neighborhoods are usually defined by first choosing a simple type of transition to obtain a new solution from a given one, and then defining the neighborhood as the set of all solutions that can be obtained from a given solution by performing one transition.

Consider now the job-shop scheduling problem. A solution corresponds to a schedule and the cost function is the maximum completion time of the schedule. In order to present a local improvement method for this scheduling problem it is useful to represent an instance by the disjunctive graph $G = (O,A,E)$.

The following neighborhood structure was proposed by van Laarhoven, Aarts and Lenstra [28]. A transition is generated by choosing two operations v and w such that: a) v and w are successive operations on the same machine M_i; b) (v, w) is a critical arc, i.e., (v, w) is in the longest path in the graph, and by reversing the order in which v and w are processed on machine M_i.

The choice is motivated by two facts. Reversing a critical arc in a directed graph can never lead to a cyclic graph [28]. Also if the reversal of a noncritical arc leads to an acyclic graph, a longest path in this new graph cannot be shorter than the longest path in the initial graph, because the new graph still contains this longest path. The size of this neighborhood is no greater than the total number of operations minus the number of machines.

To complete the design of the local improvement method to solve the job-shop scheduling problem, we need to present the methods to obtain the initial schedules. We have considered two types of methods: deterministic and randomized. For deterministic methods we can consider any method presented in the previous section. For our computational tests we use the following ones: the shifting bottleneck procedure and two priority dispatching rules with the most work remaining and the longest processing time priority function.

We are interested in randomized methods to obtain initial feasible schedules in the case that we want to repeat a certain number of runs of the local improvement method starting with different initial schedules. We have considered the following randomized methods. The first method, referred to as the 'simplest heuristic', starts by ordering the jobs randomly; then, at each time, the first operation, not yet scheduled, of the next job in this order is scheduled as soon as possible. After scheduling the current operation of the last job in the order, we start over with the next operation of the first job in the order. We repeat this until all operations have been scheduled. Note that this method is very simple, since the method just constructs a schedule without any type of optimization.

The next methods are randomized versions of the priority dispatching rules. They look like the deterministic dispatching rule but instead of selecting the operation with maximum priority, they select a random operation from the three available operations with highest priority, and schedule it as soon as possible. In case there are only two available operations, they select one of these randomly. For the computational tests, we use the same priority functions as in the deterministic case.

The next randomized method is a simplified version of the shifting bottleneck procedure, without reoptimization. At each step, a random machine is selected and then, as for the shifting bottleneck procedure, we define a one-machine scheduling problem and solve it to optimality using Carlier's algorithm [8]. The order in which the operations are processed on the respective machine is the optimal order obtained from the one-machine scheduling problem.

2.3. Simulated annealing methods

Simulated annealing has its origins in a simulated model for growing crystals, known as the Metropolis algorithm [26]. The application of the simulated annealing technique to optimization methods was proposed independently by Kirkpatrick, Gellat and Vechi [21] and Cerny [11]. Since then, this technique has gained popularity and there are many reports in the literature of applications of simulated annealing to different problems. In [19] and [20] an extensive study of the applications of simulated annealing to some well-known optimization problems is presented, as for example the graph partitioning problem and the coloring problem.

Simulated annealing accepts solutions with increased value for the cost function, called uphill moves, with small and decreasing probability, in an attempt to get away from a local optimal solution and keep exploring the region of the feasible solutions. The probability is controlled by a parameter known as the temperature, which is gradually reduced from a high value, at which most uphill moves are accepted, to a low one at which few, if any, such moves are accepted.

An important design factor of the simulated annealing methods is the cooling schedule, i.e. the scheme that controls the temperature. The scheme most commonly used is the geometric cooling rule, in which the temperature is held steady for some prespecified constant L number of trials, and is then multiplied by some given reduction factor. This provides no asymptotic guarantees, but seems to work well in practice.

To have a well-defined algorithm we have to make a variety of choices which fall into two categories: those associated with the specific problem and those that are generic [19]. The problem-specific choices are the same ones as for the local improvement methods: define the set of solutions, the cost function, the method to obtain an initial schedule and the neighborhood structure. The choices that are generic for any simulated annealing procedure are: determine an initial temperature, the reduction factor, the temperature length L and when the method stops.

Matsuo, Suh and Sullivan [25] presented an approximation method to the job-shop scheduling problem, designated as a controlled search simulated annealing. The method is a simulated annealing method that utilizes a good initial solution, a relatively small search neighborhood, and the cooling schedule is such that the acceptance probabilities of uphill moves are independent of the change in the cost function. The neighborhood used is somewhat more complicated than the one presented before, and is based on the identification of a series of operations that affect the schedule length.

Later, van Laarhoven, Aarts and Lenstra [28] presented a simulated annealing method for the job-shop scheduling problem. They use the previous neighborhood, with a cooling schedule that is more complicated than the pure geometric cooling schedule. They proved that the algorithm asymptotically converges to a global optimal

```
1. Get an initial schedule $\sigma$.
2. Get an initial temperature $T = startemp$
$L = sizefactor \times n(m-1)$, $count = 0$ and $nit = 0$.
3. While (($count < 5$) and ($nit < TIT$)):
   3.1. Perform the following loop $L$ times:
        3.1.1. Pick a random neighbor $\sigma'$ of $\sigma$.
        3.1.2. Let $\Delta = C_{max}(\sigma') - C_{max}(\sigma)$.
        3.1.3. If $\Delta \leq 0$ (downhill move), set $\sigma = \sigma'$.
        3.1.4. If $\Delta > 0$ (uphill move), set $\sigma = \sigma'$ with probability $e^{-\frac{\Delta}{T}}$.
   3.2. Set $T = tempfactor * T$ and $nit = nit + 1$.
   3.3. Let $pc$ = percentage of accepted moves in the last $L$ iterations.
   3.4. If the best schedule was changed let $count = 0$.
   3.4. Else, if ($pc < minpercent$) let $count = count + 1$.
4. Return the best solution found $\sigma_{best}$.
```

Fig. 2. Implemented simulated annealing method.

schedule and concluded that their algorithm finds good schedules but at the cost of large running times.

In our implementation of simulated annealing, see Fig. 2, we use the same methods to generate the initial schedule as for the local improvement method, namely the neighborhood structure of van Laarhoven, Aarts and Lenstra [28], but we use a different cooling schedule. Our cooling schedule is very similar to the one presented in [19] and [20]. It is a geometric cooling schedule, where the initial temperature, *startemp*, and the reduction factor, *tempfactor*, are given. The number of iterations performed at the same temperature, L, is equal to the maximum size of the neighborhood, $m(n-1)$, multiplied by a given parameter, designated by *sizefactor*. The method stops if the parameter *count* ever reaches 5, where *count* is initialized to zero and it is incremented by one each time the percentage of accepted moves in the last L iterations at the same temperature is less than or equal to a given parameter, *minpercent*, and it is reset to zero each time the best solution found so far is updated. As we did for the local improvement methods, in order to control the running time, we limited the number of repetitions of the outside loop by a given parameter TIT.

3. Large-step optimization methods

Martin, Otto and Felten [24] introduced a new class of randomized methods, designated as large-step optimization methods, where at each iteration a large step is performed, followed by a local optimization method. These methods only consider local optimal solutions and therefore the solution space is reduced. Also, they have the property of being able to make large changes in the current solution, even at low temperatures, and thus getting out of possible valleys of the cost function.

Local improvement methods proceed downhill for a while, making good progress, but they tend to get trapped in local optimal solutions, usually far away from the global optimal solution. Simulated annealing methods try to improve this by accepting uphill moves depending on a decreasing probability controlled by the temperature parameter. But, at small temperatures, they also tend to get stuck in valleys of the cost function. Large-step optimization methods allow to leave these valleys even at small temperatures, and only then a local search optimization method is applied, returning to a local optimal solution. At this point, an accept/reject test is performed. A description of a large-step optimization method is presented in Fig. 3.

The three main routines of a large-step optimization method are: the method to perform a large step, the one for the small steps and the accept/reject test. The accept/reject test can accept only downhill moves and, in this case, we designate the method by large-step local optimization. This looks like the local improvement methods presented before, but where only local optimal solutions (with respect to the small steps) are considered. On the other hand, the accept/reject test can accept all moves; we desig-

```
1. Get an initial schedule $\sigma_0$.
2. Perform the following loop numiter times:
   2.1. Large step: change $\sigma_i$ to $\sigma_{i+1}$ by performing a big perturbation.
   2.2. Small steps: run a local search optimization method
        (local improvement or simulated annealing) with initial
        solution $\sigma_{i+1}$, and obtain $\sigma'_{i+1}$.
   2.3. Perform an accept/reject test comparing $\sigma_i$ with $\sigma'_{i+1}$,
        if $\sigma'_{i+1}$ is accepted, let $\sigma_{i+1} = \sigma'_{i+1}$ .
3. Return the best solution found $\sigma_{best}$.
```

Fig. 3. Large-step optimization method.

nate this method by large-step random walk. The accept/reject test can also look like the test done in a simulated annealing method, i.e. all downhill moves are accepted and a uphill move is accepted with a small and decreasing probability. In this case, we designate the method by large-step Markov chain.

The large step should be constructed so that valleys are easily climbed over, and they should be specially tailored to the problem in consideration [24]. The methods that are usually associated with the small steps are local optimization algorithms like the local improvement method or the simulated annealing method.

As suggested by Johnson [18], these methods can be viewed as repeated runs of the local improvement or simulated annealing methods, designated as restart methods. But instead of starting with a different initial random solution at each iteration, the current solution is perturbed and used as the initial solution for the next run. In this way, all the achievements from previous runs are not totally lost in the next runs. Also, these methods have the advantage of combining the ideas associated with local optimization methods and tailored heuristics.

Different applications of the large-step optimization methods have appeared in the literature. Martin, Otto and Felten [24] developed a large-step Markov chain for the traveling salesman problem, where they used a 4-opt move as the large step, and, as the small steps, a local improvement method based on 3-opt moves. Johnson [17] also applies a large-step local optimization method to the traveling salesman problem, using a 4-opt move as the large-step and the Lin and Kernighan heuristic for the small steps. Lucks [23] applied large-step local optimization, random walk, and Markov chain to the graph partitioning problem, where the large step is done by interchanging a randomly selected set of a given size and, as small steps, the Kernighan and Lin approximation method was used.

Next, we propose the application of the large-step optimization methods to the job-shop scheduling problem. The methods used to obtain the initial schedules are the ones presented in the previous section. For the small steps, we consider the following methods: the local improvement method and the simulated annealing method presented in previous sections.

An important part of the large-step optimization methods is the large-step itself. As mentioned this procedure should make a large modification in the current schedule to drive the search to a new region and, at the same time, to perform some kind of optimization to obtain a schedule not too far away from a local (or global) optimum schedule. These two factors of the large-step method are the main differences from local search and restart methods. We propose four methods to perform the large step. The first three methods are based on choosing one or two machines and reordering the operations to be processed on these machines, using some prespecified method. When the operations of a machine are reordered, a cycle can be created, and in this case the schedule is no longer feasible. Before we present the methods for the large step we will show how cycles can be avoided.

Let σ be a complete feasible schedule for the problem, and $G_\sigma = (V, A_\sigma)$ the acyclic graph associated with this schedule. Let σ_{M_i} be a schedule obtained from σ by reordering the operations to be processed on machine M_i. The new order of the operations to be processed on M_i is obtained by some prespecified method that outputs a permutation of these operations. For example, the shifting bottleneck heuristic [2] defines one-machine problems for some machine M_i and solves these problems to optimality; a permutation of the operations to be processed on M_i is obtained. Then, construct σ_{M_i} from σ by reordering the operations of this machine by the optimal order obtained. Note that the schedule σ_{M_i} can be infeasible, i.e. there can exist a cycle in the graph $G_{\sigma_{M_i}}$.

To avoid the appearance of cycles, consider precedence relations between the operations for a given machine M_i. We write $O_{ik} \prec O_{ij}$ if O_{ik} and O_{ij} are operations to be processed on the same machine M_i and it is required that O_{ik} is completed before O_{ij} can start. We say that an edge e or the corresponding arc (i.e., the edge with an orientation) e belongs to a machine M_i, $e \in M_i$, if $e = (O_{ij}, O_{ih})$ and operations O_{ij} and

O_{ih} have to be processed on machine M_i. Let $G_\sigma \backslash M_i$ be a directed graph equal to G_σ but where all the arcs $e \in M_i$ are removed. We shall define below precedence constraints $Prec_i$ associated with M_i which guarantee that, if satisfied and while maintaining the order of the operations for the remaining machines, the complete schedule is feasible.

To obtain the precedence constraints $Prec_i$, apply the following algorithm. Consider the graph $G_\sigma \backslash M_i$ and all edges associated with the disjunctive constraints of M_i. At each step, put back exactly one edge $e = (O_{ik}, O_{ij}) \in M_i$ in $G_\sigma \backslash M_i$. Since the graph of the initial solution is acyclic, this creates at most one cycle. If a cycle is obtained, identify it and define a precedence relation between O_{ik} and O_{ij} such that this cycle will be not produced. If there is a path from O_{ik} to O_{ij}, then $O_{ik} \prec O_{ij}$. Remove this edge and consider another edge $e' \in M_i$; keep doing this until we have considered all edges belonging to M_i.

In the case that a job has more than one operation to be processed on the same machine, introduce a precedence constraint between these operations.

Theorem 1. *Let σ_{M_i} be the schedule obtained from σ by reordering the operations to be processed on machine M_i so that the above precedence relations $Prec_i$ are respected. Then σ_{M_i} is feasible, i.e. $G_{\sigma_{M_i}}$ is acyclic.*

Proof. First, we show that there cannot exist precedence relations of the form: $O_{ik} \prec O_{ij}$ and $O_{ij} \prec O_{ik}$, since, if $O_{ik} \prec O_{ij}$, then there is a path from O_{ik} to O_{ij} in $G_\sigma \backslash M_i$ and a symmetric statement can be made for $O_{ij} \prec O_{ik}$. This implies that there is a cycle in the graph G_σ, which is a contradiction. Now, we will assume that there exists a cycle and, by induction on the number of arcs in the cycle, we will arrive at a contradiction, which proves that no cycles exist in $G_{\sigma_{M_i}}$.

By the definition of the precedence relation, there is no cycle with only one arc $e_0 \in M_i$. Consider a cycle with two arcs in M_i, (O_{ik}, O_{ij}) and (O_{ih}, O_{il}). Note that these arcs can be associated with a precedence constraint or just indicate the new order of the operations. Therefore, these two cases have to be taken care of separately.

1. If $O_{ik} \prec O_{ij}$, then there is a path from O_{ik} to O_{ij} in $G_\sigma \backslash M_i$. Therefore, when the edge (O_{ih}, O_{il}) is put back in $G_\sigma \backslash M_i$, a cycle is found involving O_{il} and O_{ih}. Consequently $O_{il} \prec O_{ih}$, and O_{ih} could not be scheduled before than O_{il}. The case is similar if $O_{ih} \prec O_{il}$, or when both precedence constraints $O_{ik} \prec O_{ij}$ and $O_{ih} \prec O_{il}$ are present.

2. Suppose that (O_{ik}, O_{ij}) and (O_{ih}, O_{il}) are just arcs obtained by the new order, i.e. there are no precedence constraints between O_{ik} and O_{ij}, and between O_{ih} and O_{il}. But there is a path between O_{ij} and O_{ih} in $G_\sigma \backslash M_i$, so $O_{ij} \prec O_{ih}$ and a path can also be found between O_{il} and O_{ik}, implying $O_{il} \prec O_{ik}$. Therefore, since the new order is a permutation of the operations in M_i that respects the precedence constraints, one could not have O_{ik} processed before O_{ij} and O_{ih} processed before O_{il}, which is a contradiction.

Now, assume that there are no cycles with at most $r-1$ arcs in M_i in $G_{\sigma_{M_i}}$, and there is one with r arcs. For simplicity assume that the arcs in the cycle in M_i appear in the following order: (O_{i1}, O_{ic_1}), $(O_{i2}, O_{ic_2}), \ldots, (O_{ir}, O_{ic_r})$. Pick any of these arcs, say (O_{il}, O_{ic_l}). Now $O_{il} \prec O_{ic_l}$ and there is a path in $G_\sigma \backslash M_i$ from O_{il} to O_{ic_l} and therefore there is a cycle in $G_{\sigma_{M_i}}$ with $r-1$ arcs belonging to M_i, which is a contradiction. Now, assume that there is no precedence constraint between O_{il} and O_{ic_l}, but the following precedence constraints do exist: $O_{ic_1} \prec O_{i2}$, $O_{ic_2} \prec O_{i3}, \ldots, O_{ic_r} \prec O_{i1}$. Then, the above order is not a permutation satisfying all the precedence constraints, which is a contradiction.

Therefore, we can conclude that any method that constructs a permutation of the operations to be processed in M_i respecting the precedence constraints constructed as before can be used to give the new order of these operations, with the guarantee that the complete schedule will have no cycles. □

As mentioned before, the first three methods proposed to perform the large steps are based on obtaining new orders for the operations on one or two machines, and are presented next.

Two random machines; Carlier's algorithm: This method starts by removing all the arcs associated with two randomly chosen machines. Then it solves the one-machine scheduling problem for one of the machines using Carlier's algorithm [8] and puts back the arcs according to the new optimal order obtained. Next, it does the same for the other machine. Note that Carlier's algorithm works in the presence of precedence constraints. This method is based on a branch-and-bound method.

Two random machines; early-late algorithm: This method looks like the previous one, but instead of defining a simple one-machine scheduling problem it also considers the lags on a chain, i.e., the one-machine scheduling problem with lags on a chain [22]. Then it uses the early-late algorithm to solve this problem, [22]. Note that in this case, preemption is allowed; therefore the method does not output a permutation of the operations. To obtain such a permutation, the completion times obtained by the early-late algorithm are sorted in increasing order, and the permutation is used to obtain the new order for the operations on the given machine. The one-machine scheduling problem with lags on a chain considers only the precedence constraints between jobs that are not in the chain. Therefore, after changing the order of the operations for some machine, we check if a cycle exists. In the case that a cycle is created, this means that there is a precedence constraint between a job in the chain and one not in the chain that is violated. So, we apply the preemptive EDD rule, and again sort the completion times to obtain the new order. Observe that, by the way the release dates and the delivery times are defined, the precedence constraints are respected.

Bottleneck machine; random nonpreemptive EDD rule: This method starts by obtaining the bottleneck machine over all the machines that have at least one arc in the longest path in the acyclic graph of the current schedule. The bottleneck machine is obtained as in the shifting bottleneck procedure, i.e. it is the machine that gives the largest value when solving to optimality the associated one-machine scheduling problem. Apply the nonpreemptive EDD rule to the one-machine scheduling problem where the release dates and the processing times are obtained as usual and the delivery times are randomly generated. The precedence constraints between the operations are also considered. Note that, by the way the one-machine scheduling problem is defined, the new order will respect the precedence constraints and therefore, by the previous theorem, no cycle will appear in the new schedule.

To obtain all the precedence constraints in the first and third methods above to perform a large step, we need to make $n(n-1)$ calls of the method to check for a cycle. To check for a cycle in a digraph associated with a schedule takes $O(nm)$ since these graphs have $2n+(m-1)n+(n-1)m$ arcs. The method to check for a cycle is a depth-first-search method [3].

Since we have observed that a cycle is rarely encountered, if the precedence constraints are ignored when we apply these methods, we decide to use a different strategy: Ignore the precedence constraints, apply the method and then check for cycles. If a cycle is found, identify the violated precedence constraint and repeat; keep repeating until no more cycles exist. This strategy was also applied by Adams, Balas and Zawack [2] since they also observed that the appearance of cycles is rare.

The last method proposed to perform the large step is different. This method is based on Matsuo, Suh and Sullivan's neighborhood structure [25] which interchanges consecutive operations. The method is presented next:

1. Let σ be a schedule and let G_σ be the associated digraph.
2. Pick a random arc (O_{iu}, O_{iv}) in the longest path in G_σ.
3. Interchange O_{iu} and O_{iv}.
4. Let $\tilde{O}_u = O_{iu}$.
5. While $\tilde{O}_u$ is not the first operation of job J_u:
(a) Obtain the previous operation of job J_u, say O_{ku}.
(b) If O_{ku} is not the first operation to be processed on machine M_k, obtain the operation that immediately precedes O_{ku} on M_k, and inter-

change these two operations if a cycle is not created.

(c) Let $\tilde{O}_u = O_{ku}$.

6. Let $\tilde{O}_v = O_{iv}$.

7. While $\tilde{O}_v$ is not the last operation of job J_v:

(a) Obtain the next operation of job J_v, say O_{lv}.

(b) If O_{lv} is not the last operation to be processed in machine M_l, obtain the operation that immediately follows O_{lv} in M_l, and interchange these two operations if a cycle is not created.

(c) Let $\tilde{O}_v = O_{lv}$.

Note that in the above method, both loops terminate, since the number of operations of each job is finite.

All the methods perform a large modification in the current schedule and, specially the first two, optimize parts of the schedule.

4. Computational results

All the methods were implemented in C and all tests were run on a Sparcstation 1. The running times are given in seconds.

For our experiments we consider the following instances: MT10, the 10-job 10-machine instance of the job-shop scheduling problem proposed in [15], ABZ5 ($n = 10, m = 10$) and ABZ8 ($n = 20, m = 15$) by Adams, Balas and Zawack, ORB1 ($n = 10, m = 10$) by Applegate and Cook, and LA19 ($n = 10, m = 10$), LA20 ($n = 10, m = 10$), LA21 ($n = 15, m = 10$), LA27 ($n = 20, m = 10$), LA39 ($n = 15, m = 15$) by Lawrence and CAR5 ($n = 10, m = 6$) by Carlier. All the data was generously given by David Applegate and Bill Cook. The length of the optimal schedules for instances LA21, ABZ8 and LA27 is not known and the best lower bound that we are aware of is presented in Table 12. For the remaining ones the optimal length is presented in the same table. Whenever a method obtains this value for these last instances it is indicated by *.

In all the following tables, the first row for each instance indicates the length of the best schedule obtained, and the second row, if present, indicates the running time in seconds. If any result is not known, the respective entry is left blank.

We use the following notation and indicate the section where the method is described:

- DR(MWR): dispatching rule (most work remaining) (2.1);
- DR(LPT): dispatching rule (longest processing time) (2.1);
- SS: simplest schedule (2.2);
- RDR(MWR): random dispatching rule (most work remaining) (2.2);
- RDR(LPT): random dispatching rule (longest processing time) (2.2);
- RSBH: random simplest shifting bottleneck heuristic (2.2);
- SBH: shifting bottleneck heuristic (2.1);
- 2mach/BB: large-step procedure two random machines, Carlier's algorithm (3);
- 2mach/EL: large-step procedure two random machines, early-late algorithm (3);
- bot/EDD: large-step procedure bottleneck machine, random nonpreemptive EDD rule (3);
- MSS: large-step based on the Matsuo, Suh and Sullivan neighborhood (3);
- LI: local improvement optimization method (2.2);
- SA: simulated annealing (2.3);
- LSLO: large-step local optimization (3);
- LSMC: large-step Markov Chain (3);
- LSRW: large-step random walk (3);
- $C^*_{\max} | LB$: Length of the optimal schedule, or, if this value is not known we present a lower bound.
- NSBH: New improved shifting bottleneck heuristic [5]
- SHUF: Shuffle heuristic [4]
- TS: Tabu search method [13]

In Section 4.1 we present the results of some tailored, deterministic and random, methods to obtain initial schedules to be used by the local search and large-step optimization methods. We present in Tables 1 and 2 the lengths of the schedule and the running time obtained by the deterministic and random dispatching rules (most work remaining and longest processing time), the simplest heuristic, the shifting bottleneck heuristic and the random simplest shifting bottleneck heuristic.

In Section 4.2 we study the different combinations between the proposed methods to perform the small and large steps in the large-step optimization methods presented in Section 3. In Tables 3 and 4 we present the results when we apply the large-step local optimization relative to the 4 different large-steps (2mach/BB, 2mach/EL, bot/EDD and MSS) and considering as the small-step method the local improvement and simulated annealing, respectively. In Tables 5 and 6 we present the results related with the large-step random walk and Markov chain, respectively, for the four different large-step and simulated annealing as the small-step. In Table 7 with consider the three variants of the large-step optimization, local optimization, random walk and Markov chain with large-step 2mach/BB and small-step simulated annealing, running each one fewer times than in previous examples.

In Section 4.3 we compare the different local search and large-step optimization methods. In Tables 8 and 9 we present the results obtained by each of the following methods: local improvement, simulated annealing and the three variants of the large-step optimization, local optimization, random walk and Markov chain with large-step 2mach/EL and small-step simulated annealing, starting with a initial scheduled obtained by the shifting bottleneck heuristic and dispatching rule (most work remaining), respectively. In Tables 10 and 11 we consider several trials of the same methods just mentioned, but considering the initial scheduled obtained by the shifting bottleneck heuristic and the simplest heuristic, respectively.

The conclusions of each study are presented at the end of each section respectively. Later on we will compare some of our results with others known from the literature.

4.1. Initial schedule methods

Next we present the results obtained when we run the methods to obtain an initial schedule. We consider the methods described in Sections 2.1 and present the length of the schedules obtained and the running times in Table 1. In Table 2 we present the results when we repeat the random methods roughly the time taken by the shifting bottleneck heuristic for each instance.

We can observe that the shifting bottleneck heuristic performs better than any other method to obtain an initial schedule. But it takes more time than a single run of any other method. Even when we repeat the random methods for the same amount of time that one run of SBH takes, SBH still gives the best results for all the instances except one, CAR5. The DR(LPT) and RDR(LPT) performs poorly on any instance.

Table 1
Lengths for the initial schedules and the running times (one trial); *study: Initial schedules*

Instance	DR(MWR)	DR(LPT)	SS	RSBH
MT10	1289	2851	1322	1129
	5.0×10^{-2}	3.3×10^{-2}	3.3×10^{-2}	2.0×10^{-1}
ABZ5	1451	4591	1496	1451
	1.7×10^{-2}	1.7×10^{-2}	3.3×10^{-2}	2.3×10^{-1}
LA19	1188	3180	1076	986
	6.7×10^{-2}	6.6×10^{-2}	3.3×10^{-2}	2.1×10^{-1}
LA20	1404	3242	1274	1048
	3.3×10^{-2}	3.3×10^{-2}	5.0×10^{-2}	1.8×10^{-1}
ORB1	1626	2708	1667	1325
	5.0×10^{-2}	6.6×10^{-2}	5.0×10^{-2}	2.0×10^{-1}
LA21	1494	5471	1428	1253
	6.7×10^{-2}	6.7×10^{-2}	3.3×10^{-2}	3.2×10^{-1}
ABZ8	1102	4696	923	823
	1.2×10^{-1}	8.3×10^{-2}	1.2×10^{-1}	8.5×10^{-1}
LA27	1918	6218	1702	1442
	6.7×10^{-2}	6.7×10^{-2}	6.0×10^{-2}	4.5×10^{-1}
CAR5	15189	13626	9645	8832
	3.3×10^{-2}	6.7×10^{-2}	3.3×10^{-2}	6.7×10^{-2}
LA39	1778	6175	1810	1581
	8.3×10^{-2}	8.3×10^{-2}	6.7×10^{-2}	6.0×10^{-1}

Table 2
Lengths for the initial schedules and the running times (repeated trials); *study: Initial schedules*

Instance	SBH	SS	RDR(MWR)	RDR(LPT)	RSBH
MT10	952	1224	1199	1993	1129
ABZ5	1270	1364	1394	2569	1415
LA19	863	1003	1002	1854	986
LA20	918	1065	1221	1793	1048
ORB1	1176	1436	1543	1996	1325
LA21	1128	1333	1340	3034	1253
ABZ8	738	851	964	2600	823
LA27	1353	1574	1727	3683	1442
CAR5	8873	7862	13134	12954	8832
LA39	1301	1564	1695	3683	1581

Therefore, in what follows we will not consider these methods anymore. The simplest heuristic (SS) performs quite well on most of the instances, specially considering that this heuristic just starts by ordering the jobs randomly; then, at each time, the first operation, not yet scheduled, of the next job in this order is scheduled as soon as possible. Note that this method performs better than the deterministic and randomized dispatching rules in most of the cases. The randomized shifting bottleneck heuristic (RSBH) also gave good results considering that it runs faster than the SBH.

4.2. Small / large step combinations

In this section we try to identify the best combination large-step/small-step for the large step optimization methods. We run one trial of all the large-step optimization methods starting with the schedule obtained by SBH. For the different versions in which the local improvement methods is used as the small-step method, we run at most 1000 iterations of the large step optimization method. Since one run of the simulated annealing method takes considerably more time than one of the local improvement methods, when the simulated annealing is used as the small-step method we just run 100 iterations. As concluded in [22] the best results where obtained when *startemp* = 500, *tempfactor* = 0.99, *minpercent* = 0.02, *sizefactor* = 2 and *tit* = 1000, for almost all the instances tested. For more details about the experiment that we used to set the parameters of the simulated annealing method (*startemp,tempfactor, minpercent* and *sizefactor*) and results see [22].

The methods to perform the large step are: two random machines, Carlier's algorithm (2mach/BB); two random machines, early-late algorithm (2mach/EL); bottleneck machine, random nonpremptive EDD rule (bot/EDD) and the Matsuo, Suh and Sullivan method (MSS). Any of these methods performs large changes in a schedule, at any iteration of the method.

In Table 3 we present the results for the large-step local optimization when the local improvement method was used to perform the small-steps. For any large-step method, these

Table 3
Large-step local optimization; small-step: local improvement

	Large-step			
	2mach/BB	2mach/EL	bot/EDD	MSS
MT10	949	949	949	949
	1.10×10^2	1.31×10^2	3.65×10^2	2.93×10^2
ABZ5	1258	1258	1260	1258
	1.19×10^2	1.40×10^2	2.57×10^2	2.35×10^2
LA19	863	863	863	863
	9.00×10^1	1.25×10^2	1.58×10^2	2.58×10^2
LA20	918	918	918	918
	9.20×10^1	1.22×10^2	2.08×10^2	2.12×10^2
ORB1	1170	1161	1152	1170
	1.18×10^2	1.68×10^2	2.74×10^2	2.61×10^2
LA21	1120	1115	1120	1116
	1.87×10^2	2.42×10^2	5.09×10^2	4.69×10^2
ABZ8	706	719	734	722
	5.03×10^2	7.55×10^2	7.67×10^2	1.31×10^3
LA27	1270	1290	1347	1315
	3.19×10^2	5.42×10^2	1.32×10^2	6.84×10^2
CAR5	8831	8831	8806	8831
	7.30×10^1	7.50×10^1	1.21×10^2	6.18×10^1
LA39	1286	1268	1301	1275
	3.08×10^2	4.19×10^2	5.23×10^2	9.11×10^2

small-step methods fail to obtain a good local optimum; it gets stuck very soon in a valley. There is not much difference in the results obtained from different large-steps methods.

Since the same type of results were obtained for the other methods, large-step random walk and Markov-chain, we will not present the results obtained by these methods, we refer the reader to Lourenço [22] for more information. Also, the running time obtained by these methods for all instances were very similar to the ones obtained to the large-step local optimization method, we will just present the running times for this one. Note that when local improvement was used as the small-step methods the running time was smaller than when simulated annealing was used, but the quality of the solution obtained was also inferior.

In Table 4 we consider the large-step local optimization method where the simulated annealing is used to perform the small-steps, with the following values for the parameters: *startemp* = 500, *tempfactor* = 0.95, *minpercent* = 0.02, *tit* = 100 and *sizefactor* = 1. Using simulated annealing we obtained better results than when we used the

Table 4
Large-step local optimization; small-step: simulated annealing

	Large-step			
	2mach/BB	2mach/EL	bot/EDD	MSS
MT10	949	937	952	952
	5.66×10^3	5.80×10^3	5.64×10^3	2.72×10^3
ABZ5	1239	1245	1240	1238
	5.69×10^3	5.67×10^3	5.69×10^3	2.65×10^3
LA19	848	842 *	842 *	850
	5.59×10^3	5.55×10^3	5.53×10^3	2.62×10^3
LA20	907	907	902 *	907
	5.67×10^3	5.63×10^3	5.59×10^3	2.64×10^3
ORB1	1100	1103	1113	1102
	5.80×10^3	5.83×10^3	5.68×10^3	2.71×10^3
LA21	1065	1072	1064	1066
	1.33×10^4	2.70×10^4	3.62×10^4	2.80×10^4
ABZ8	697	708	699	704
	2.70×10^4	2.70×10^4	3.62×10^4	2.80×10^4
LA27	1269	1265	1273	1273
	2.75×10^4	2.49×10^4	1.70×10^4	1.16×10^4
CAR5	7702 *	7882	7767	7790
	1.97×10^3	2.03×10^3	2.11×10^3	9.39×10^2
LA39	1252	1267	1251	1242
	1.98×10^4	2.98×10^4	1.94×10^4	1.38×10^4

local improvement method as the small-steps, but also it took more time. For the instances LA19, LA20 and CAR5, an optimal schedule was obtained. The best values are spread out between the different large-step methods.

In Table 5 we present the results obtained by the large-step random walk when the small-steps method was the simulated annealing. As for the large-step local optimization method, we obtained better results in this case than when we used the local improvement method. Also, the best results were obtained when the large-step method was the 2mach/BB and 2mach/EL.

Table 5
Large-step random walk; small-step: simulated annealing

	Large-step			
	2mach/BB	2mach/EL	bot/EDD	MSS
MT10	951	951	952	952
ABZ5	1244	1234 *	1236	1238
LA19	842 *	842 *	850	843
LA20	902 *	907	902 *	907
ORB1	1083	1101	1103	1106
LA21	1072	1072	1079	1065
ABZ8	703	702	704	706
LA27	1264	1264	1270	1270
CAR5	7734	7808	7821	7720
LA39	1261	1258	1250	1251

Table 6
Large-step Markov chain; small-step: simulated annealing

	Large-step			
	2mach/BB	2mach/EL	bot/EDD	MSS
MT10	947	949	949	949
ABZ5	1242	1240	1238	1239
LA19	842 *	850	846	848
LA20	907	902 *	907	907
ORB1	1092	1101	1103	1092
LA21	1061	1058	1059	1071
ABZ8	705	704	698	709
LA27	1270	1264	1269	1269
CAR5	7761	7725	7822	7720
LA39	1264	1256	1256	1253

The results obtained by the large-step Markov chain are presented in Table 6, using as the small-steps method the simulated annealing method. The parameters for the simulated annealing method were set up as in the runs for the other large-step methods. For the large-step Markov chain method, the start temperature and the temperature factor were 500 and 0.99, respectively. To obtain these values we run the large-step Markov chain method for four different sets of values for the start temperature and temperature factor. The best results were obtained for the values indicated, but there was not a significant difference between the results.

Overall the proposed methods to perform the large-step, the best results were obtained for the large-step method 2mach/BB and 2mach/EL.

In Table 7 we present the results when we applied the large-step optimization methods using the simulated annealing method and the 2mach/BB, where in this case we run just 10 iterations of the method but we let run the simulated annealing method for 1000 iterations. Comparing these results with the respective ones in Tables 4, 5 and 6, we can observe that by running fewer iterations of the large-step optimizations methods and more of the simulated annealing method we did not get better results.

Table 7
Large-step optimization with few large-steps using simulated annealing/2mach/BB

	LSLO	LSRW	LSMC
MT10	947	952	952
ABZ5	1249	1244	1250
LA19	856	854	861
LA20	907	911	914
ORB1	1145	1131	1105
LA21	1065	1096	1085
ABZ8	712	707	710
LA27	1279	1299	1279
CAR5	8047	8047	7821
LA39	1261	1294	1261

4.3. Comparison of local optimization methods

In this section, we compare the performance of the following methods: local improvement, simulated annealing with best parameters, large-step local optimization, random walk and Markov chain with best parameters. The comparison is carried out by applying the algorithms to 10 instances of the job-shop scheduling problem. Initially, we performed one trial of each method starting with a schedule obtained by a deterministic method.

The values of the parameters for the simulated annealing method were as follows: *startemp* = 500, *tempfactor* = 0.99, *minpercent* = 0.02, *sizefactor* = 4 and *tit* = 5000. For the large-step optimization methods we use as the large-step method the two random machines/early-late algorithm, and as the small-steps method the simulated annealing with the same parameters as above, except for *sizefactor* = 2 and *tit* = 1000. We run 100 iterations of this method. For the large-step Markov chain method, we consider the start temperature to be 500 and the temperature factor to be 0.99.

In Tables 8 and 9 we present the results obtained, when the initial schedule was obtained by the shifting bottleneck heuristic and the dispatching rule (most work remaining), respectively. The local improvement method did not improve the initial solution, when this one was obtained by the shifting bottleneck heuristic, except for instance ABZ8, where 1 move was accepted. Overall, the local improvement method is clearly infe-

Table 8
Approximation algorithms; one trial; initial schedule: shifting bottleneck heuristic

	LI	SA	LSLO	LSRW	LSMC
MT10	952	949	944	937	941
	2.76×10^0	1.08×10^4	6.59×10^4	1.01×10^5	9.86×10^4
ABZ5	1270	1246	1239	1236	1234 *
	2.40×10^0	1.09×10^4	6.12×10^4	8.96×10^4	9.22×10^4
LA19	863	843	842 *	842 *	842 *
	2.98×10^0	1.08×10^4	5.21×10^4	7.69×10^4	8.09×10^4
LA20	918	902 *	902 *	902 *	902 *
	3.03×10^0	9.78×10^2	5.21×10^4	8.29×10^4	8.38×10^4
ORB1	1176	1101	1078	1083	1059 *
	2.73×10^0	1.07×10^4	6.42×10^4	1.04×10^5	1.00×10^5
LA21	1128	1059	1048	1053	1047
	4.53×10^0	2.47×10^4	1.43×10^5	2.40×10^5	2.49×10^5
ABZ8	736	679	682	681	680
	2.97×10^1	1.03×10^5	5.56×10^5	5.01×10^5	5.80×10^5
LA27	1353	1287	1242	1254	1254
	8.38×10^0	4.68×10^4	2.63×10^5	4.90×10^5	4.87×10^5
CAR5	8873	8308	7720	7702 *	7720
	3.66×10^{-1}	4.08×10^3	2.67×10^4	3.85×10^4	3.83×10^4
LA39	1301	1244	1239	1239	1240
	1.78×10^1	5.59×10^3	2.68×10^5	4.89×10^5	2.33×10^5

Table 9
Approximation algorithms; one trial; initial schedule: dispatching rule (most work remaining)

	LI	SA	LSLO	LSRW	LSMC
MT10	1130	944	945	941	940
	2.83×10^{-1}	1.06×10^4	8.16×10^4	9.96×10^4	5.60×10^4
ABZ5	1322	1242	1263	1236	1234 *
	2.16×10^{-1}	1.08×10^4	7.51×10^4	9.38×10^4	5.27×10^4
LA19	993	842 *	842 *	842 *	842 *
	2.66×10^{-1}	1.08×10^4	6.29×10^4	7.16×10^4	4.00×10^4
LA20	1055	907	902 *	902 *	902 *
	6.16×10^{-1}	1.09×10^4	6.32×10^4	7.96×10^4	4.07×10^4
ORB1	1411	1098	1078	1074	1089
	6.67×10^{-1}	1.07×10^4	7.96×10^4	9.52×10^4	5.88×10^4
LA21	1255	1065	1048	1058	1053
	7.33×10^{-1}	2.19×10^4	1.96×10^5	2.53×10^5	1.57×10^5
ABZ8	901	695	688	682	683
	2.30×10^0	8.73×10^4	6.03×10^5	6.04×10^4	6.08×10^5
LA27	1518	1260	1255	1255	1256
	1.81×10^0	6.86×10^3	2.71×10^5	4.87×10^5	3.20×10^5
CAR5	11058	7832	7720	7727	7702 *
	3.00×10^{-1}	4.29×10^3	3.08×10^4	3.93×10^4	3.93×10^4
LA39	1560	1264	1242	1246	1242
	1.73×10^0	4.71×10^4	2.44×10^5	4.93×10^5	3.09×10^5

rior to the other methods, when the running time is not taken in consideration.

The large-step optimization methods outperformed the simulated annealing method, but they also take more computational time. The large-step optimization methods obtained several optimal solutions, for the problems where this value is known. For the remaining ones, these methods obtained good solutions, some of them are better than the best known feasible solutions obtained by approximation methods, as for example for instances LA21, LA27, and ABZ8, see Table 12. But, for the famous MT10, the optimal solution was never obtained.

When we start the full set of methods with an inferior initial solution, the local improvement method fails to obtain a good local optimal solution, but for the simulated annealing method this was not the case, in 7 out of 10, the solution obtained in Table 11 was better than the one in Table 10, for the SA column. However, for the large-step optimization methods, the use of a good initial solution gives some help, but it is not as significatively different as for the local improvement.

Between the different large-step optimization methods, there was not clear evidence that one method outperformed the remaining ones, since the results obtained by the three methods were very close. But, the large-step Markov chain method gave quite a few good results, including more optimal schedules than the remaining ones. Therefore, some randomization helps in the large-step optimization methods.

Next, we repeat the above but take into consideration the running time and random initial schedules. All methods are allowed an almost

Table 10
Approximation algorithms; several trials; initial schedule: shifting bottleneck heuristic

	SA	LSLO	LSRW	LSMC
MT10	951	937	938	939
ABZ5	1239	1239	1238	1236
LA19	843	842 *	842 *	842 *
LA20	907	902 *	902 *	902 *
ORB1	1085	1079	1078	1079
LA21	1054	1049	1055	1056
ABZ8	679	680	680	674
LA27	1264	1258	1257	1256
CAR5	7824	7821	7702	7749
LA39	1250	1242	1247	1240

Table 11
Approximation algorithms; several trials; initial schedule: simplest heuristic

	LI	SA	LSMC
MT10	1021	948	944
ABZ5	1269	1239	1236
LA19	874	842 *	842 *
LA20	914	902 *	902 *
ORB1	1203	1086	1086
LA21	1131	1054	1049
ABZ8	752	682	681
LA27	1379	1254	1254
CAR5	7749	7832	7731
LA39	1366	1248	1242

equal amount of running time, i.e. we repeat the local improvement and simulated annealing so that these methods take about the same time as one run of the large-step optimization methods. The parameters were fixed as in the previous experiment. The results are presented in Tables 10 and 11. The local improvement method was not considered when the initial schedule was obtained by the shifting bottleneck heuristic. Also, when the randomized simplest heuristic was used to obtain an initial schedule, repeated trials of the local improvement method performed poorly when compared to the remaining methods. The repeated simulated annealing method came closer to the large-step optimization methods than just one trial of the first method. However, we can observe in Tables 10 and 11 that in general the large-step optimization methods performed better.

5. Conclusions

We considered different approximation algorithms to solve the job-shop scheduling problem. We started by describing some tailored algorithms as the dispatching rules [12], and the shifting bottleneck heuristic [2], and then we presented a randomized version of these methods. Among the above methods the shifting bottleneck heuristic gave the best results when applied to 10 selected instances of the job-shop scheduling problem of different sizes.

All these methods were used to obtain an initial schedule for another set of approximation algorithms based on local optimization techniques. We review two important methods: local improvement and simulated annealing. We proposed a generalization of these local optimization methods, the large-step optimization methods. For this set of approximation algorithms, different small-step and large-step methods were proposed. We can conclude that when the simulated annealing method was used as the small-steps methods, better results were obtained than with other methods, since at each iteration of the large-step method a better local optimum is obtained. For the large-step method, the 2mach/BB and 2mach/EL gave better results in general than the remaining methods while taking about the same amount of computation time. Note that these two methods are more tailored to the problem than the remaining ones, which change the schedule without any form of partial optimization. Moreover, it is important to realize that these methods make big changes in the current schedule at any iteration of the large-step optimization methods. Therefore, the two features of these methods, optimization and big changes, play an important role in a good large-step method.

Next, we compare the performance of the following methods: local improvement, simulated annealing and the large-step optimization methods. First, we run one iteration of each method and later we allow almost the same running time for all methods. From the computational results we can conclude that the local improvement method is clearly inferior, even when several trial are performed. The large-step optimization methods outperformed the simulated annealing method, but the difference between these two came closer when the same amount of time was allowed. The large-step optimization methods found an optimal schedule more frequently than other methods. However, these methods and the simulated annealing method take a considerable amount of time, which is clearly a disadvantage. We consider this disadvantage to be compensated by the high quality of the solutions obtained and the simplicity of both methods, the simulated annealing and the large-step optimization method.

Table 12
Summary of the results

n×m	C^*_{max}|LB	SBH	NSBH	SHUF	TB	SA	LSO
MT10	930*	952	940	930*	935	944	937
ABZ5	1234*	1270	1258	1245	1236	1238	1234*
LA19	842*	863	878	848	842*	842*	842*
LA20	902*	918	922	911	902*	902*	902*
ORB1	1059*	1176	1121	1070	1064	1068	1059*
LA21	1040	1128	1071	1053	1048	1053	1047
ABZ8	635	738	708	687	678	679	674
LA27	1235	1353	1272	1269	1242	1254	1242
CAR5	7702*	8873				7720	7702*
LA39	1233*	1301	1278		1242	1244	1239

Recently, we have been aware of a successful application of tabu search to the job-shop scheduling problem [13]. This method gave very good results, in small amounts of time. Note that a big percentage of the running time for a large-step optimization method is spent by the simulated annealing method. Therefore, an improvement for large-step optimization methods should be achieved by using small-step methods that give good local optimal solutions and run in a smaller amount of time than the simulated annealing. By the above observations the tabu search methods looks like a good candidate to be used instead of the local improvement method, that are fast but gives poor local optimal solutions, and instead of simulated annealing that gives good local optimal solutions, but at very high computational cost.

To conclude, we present in Table 12 the best results obtained by simulated annealing, large-step optimization methods and the following methods: the shifting bottleneck heuristic, [2], the new version of this method [5], the tabu search method [13], and the shuffle algorithm [4]. From Table 12 we can conclude that our implementation of the simulated annealing and the large-step optimization methods outperformed the remaining methods, obtaining several optimal schedules. However, as mentioned before, they take a large amount of time.

In terms of future research, there is a need to develop theoretical results associated with the large-step optimization methods, as it has been done for the simulated annealing.

Acknowledgments

The author is grateful to David Shmoys, G. Finke and an anonymous referee for many comments and suggestions which have considerably improved this paper.

References

[1] Aarts, E.H.L., van Laarhoven, P.J.M., Lenstra, J.K., and Ulder, N.J.L., "A computational study of local search algorithms for job shop scheduling", *ORSA Journal on Computing* 6/2 (1994) 118–125.

[2] Adams, J., Balas, E., and Zawack, D., "The shifting bottleneck procedure for the job-shop scheduling problem", *Management Science* 34/3 (1988) 391–401.

[3] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., *Data Structures and Algorithms*, Addison-Wesley, MA, Reading, 1983.

[4] Applegate, D., and Cook, W., "A computational study of the job-shop scheduling problem", ORSA *Journal on Computing* 3/2 (1991) 149–156.

[5] Balas, E., Lenstra, J.K., and Vazacopoulos, A., "One machine scheduling with delayed precedence constraints", Technical Report, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1992.

[6] Barnes, J.W., "Solving the job shop scheduling problem using tabu search", Technical Report, Graduate Program in Operations Research, The University of Texas at Austin, 1991.

[7] Brucker, P., Jurish, B., and Sievers, B., "A branch & bound algorithm for the job-shop scheduling problem", *Discrete Applied Mathematics* 49/1-3 (1994) 107–127.

[8] Carlier, J., "The one-machine sequencing problem", *European Journal of Operational Research* 11 (1982) 42–47.

[9] Carlier, J., and Pinson, E., "An algorithm for solving the job-shop problem", *Management Science* 35/2 (1989) 164–176.

[10] Carlier, J., and Pinson, E., "A practical use of Jackson's preemptive schedule for solving the jobshop problem", *Annals of Operations Research* 26 (1990) 260–287.

[11] Cerny, V., "A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm", *Journal of Optimisation Theory and Applications* 45 (1985) 41–51.

[12] Conway, R.W., Maxwell, W.L., and Miller, L.W., *Theory of Scheduling*, Addison-Wesley, Reading, MA, 1967.

[13] Dell' Amico, M., and Trubian, M., "Applying tabu-search to the job-shop scheduling problem", *Annals of Operations Research* 4 (1993) 231–252.

[14] Dorndorf, U., and Pesch, E., "Evolution based learning in a job shop scheduling environment", 1992.

[15] Fisher, H., and Thompson, G.L., "Probabilistic learning combinations of local job-shop scheduling rules", in: J.

Muth and G. Thompson (eds.) *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, NY, 1963, 225–251.

[16] Garey, M.R., Johnson, D.S., and Sethi, R., "The complexity of flowshop and jobshop scheduling", *Mathematics of Operations Research* 1 (1976) 117–129.

[17] Johnson, D.S. "Local optimization and the traveling salesman problem", in: *Proc. 17th Colloquium on Automata, Languages and Programming*, Springer-Verlag, Berlin, 1990, 446–461.

[18] Johnson, D.S., "Random starts for local optimization, 1993", DIMACS Workshop on Randomized Algorithms for Combinatorial Optimization, 1993.

[19] Johnson, D.S., Aragon, C.R., McGeoch, L.A., and Schevon, C., "Optimization by simulated annealing; An experimental evaluation; part I, graph partitioning", *Operations Research* 37/6 (1989) 865–892.

[20] Johnson, D.S., Aragon, C.R., McGeoch, L.A. and Schevon, C., "Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning", *Operations Research* 39/3 (1991) 378–406.

[21] Kirkpatrick, S., Gellatt, C.D., and Vecchi, M.P., "Optimization by simulated annealing", *Science* 220/4598 (1983) 671–680.

[22] Lourenço, H.R. "A computational study of the job-shop and the flow-shop scheduling problems", Ph.D. Thesis, School of OR& IE, Cornell University, Ithaca, NY, 1993.

[23] Lucks, V.B.F., "Large-step local optimization for the graph partitioning problem", Master's Thesis, School of OR&IE, Cornell University, Ithaca, NY, 1992.

[24] Martin, O., Otto, S.W., and Felten, E.W. "Large-step markov chains for TSP incorporating local search heuristics", *Operations Research Letters* 11 (1992) 219–224.

[25] Matsuo, H., Suh, C.J., and Sullivan, R.S., "A controlled search simulated annealing method for the general jobshop scheduling problem", Technical Report 03-04-88, Dept. of Management, The University of Texas at Austin, 1988.

[26] Metropolis, W., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E., "Equation of state calculations by fast computing machines", *Journal of Chemical Physics* 21 (1953) 1087–1092.

[27] Roy, B., and Sussmann, B., "Les problèmes d'ordonnancement avec contraintes disjonctives", Notes DS no. 9 bis, SEMA.

[28] van Laarhoven, P.J.M., Aarts, E.H.L., and Lenstra, J.K., "Job shop scheduling by simulated annealing", *Operations Research* 40/1 (1992) 113–125.