J Heuristics (2016) 22:587–611
DOI 10.1007/s10732-014-9279-5

An iterated greedy metaheuristic for the blocking job shop scheduling problem

Marco Pranzo · **Dario Pacciarelli**

Received: 29 October 2013 / Revised: 17 June 2014 / Accepted: 27 December 2014 /
Published online: 18 January 2015

Abstract In this paper we consider a job shop scheduling problem with blocking (BJSS) constraints. Blocking constraints model the absence of buffers (zero buffer), whereas in the traditional job shop scheduling model buffers have infinite capacity. There are two known variants of this problem, namely the blocking job shop scheduling with swap allowed (BWS) and the one with no swap allowed (BNS). This scheduling problem is receiving an increasing interest in the recent literature, and we propose an Iterated Greedy (IG) algorithm to solve both variants of the problem. IG is a metaheuristic based on the repetition of a destruction phase, which removes part of the solution, and a construction phase, in which a new solution is obtained by applying an underlying greedy algorithm starting from the partial solution. A comparison with recent published results shows that the iterated greedy algorithm outperforms other state-of-the-art algorithms on benchmark instances. Moreover it is conceptually easy to implement and has a broad applicability to other constrained scheduling problems.

Keywords Job shop scheduling problem · Blocking · Swap · Iterated greedy · Alternative graph

1 Introduction

The job shop scheduling problem is one of the most studied problems in combinatorial optimization. In this problem, a set of jobs must be processed on a set of machines

M. Pranzo (✉)
Università degli Studi di Siena, Via Roma 56, 53100 Siena, Italy
e-mail: pranzo@diism.unisi.it

D. Pacciarelli
Università degli Studi Roma Tre, Via della Vasca Navale 79, 00146 Rome, Italy
e-mail: pacciarelli@dia.uniroma3.it

and 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. Each machine can host only one job at a time, and whenever two jobs require the same machine, a processing order must be defined between the incompatible operations. The problem consists of scheduling all operations on all machines to minimize the completion time of all jobs. The classical version of the problem assumes that jobs can be stored in a buffer of infinite capacity between consecutive operations. However, there are many practical situations in which the buffer capacity has to be taken into account. This is the case, for example, when there is no buffer to store work-in-progress or it is not convenient to store jobs due to economic considerations. Among industrial applications from the literature in which these situations occur we cite the production of concrete (Grabowski and Pempera 2000) and steel (Pacciarelli 2004; Pacciarelli and Pranzo 2004), the management of robotic systems (Levner et al. 1997), as well as surgical case scheduling (Pham and Klinkert 2008), container handling (Meersmans 2002), and railway traffic management (D'Ariano et al. 2007, 2010).

The blocking constraint models the complete absence of storage capacity between consecutive machines (Hall and Sriskandarajah 1996). In this setting, a machine is released by a job only when the subsequent machine is available for processing it. When the subsequent machine is unavailable the job remains on the current machine, thus blocking it, even after its operation is completed. For example, in railway traffic management one can view a train as a job and a rail track segment between two signals as a machine that can host at most one job at a time. A blocking constraint arises since each train, having reached the end of a track segment can enter the next segment only if this is empty. Otherwise it must wait, thus preventing other trains from entering its current segment. Therefore, regulating the traffic on a train networks corresponds to scheduling jobs on blocking machines.

Two versions of the job shop scheduling problems with blocking can be distinguished: the blocking with swap (BWS) and the blocking no-swap (BNS) versions. The need to swap operations between machines arises whenever there is a circular set of blocking operations in which each operation is waiting for a machine occupied by another operation in the set. The only way to solve such situation is that all operations in the set *swap*, i.e., move *simultaneously* to their subsequent machines so that all corresponding subsequent operations can start at the same time. In other words, a circular set of mutually blocked operations can occur in the BWS case, while it must be avoided in the BNS case, when swap is not allowed. Railway traffic management is one example of the BNS case, since a train can start moving to the next segment only after that this has been emptied. Mascis and Pacciarelli (2002) show that the problem of deciding whether a feasible schedule exists, when a certain partial schedule is given, is polynomially solvable for the BWS case and it is NP-complete for the BNS version, while the BJSS problem of finding the minimum completion time is NP-hard in both cases.

Despite the practical relevance of the BJSS problem, only in the last years few algorithmic contributions on job shop scheduling problems with blockings have been proposed in the literature. One successful approach is based on the alternative graph formulation (Mascis and Pacciarelli 2002), which allows to model the BJSS problem in both variants, with and without swaps. In Mascis and Pacciarelli (2002) a branch and bound algorithm is proposed to sove the problem exploiting the generalization to the blocking case of some properties holding for the traditional JSS problem with infinite buffer capacity. In Pranzo et al. (2003) a family of heuristics (AGH) extending the ones introduced in Mascis and Pacciarelli (2002) has been proposed and tested. Based on the AGH heuristics, in Meloni et al. (2004) a rollout algorithm is proposed for solving four variants of the job shop scheduling problem including, among the others, the BWS and the BNS versions. In Liu and Kozan (2011), a constructive algorithm with local search improvement is proposed for solving the blocking job shop scheduling problem with parallel-machines and applied to solve a train scheduling problem. Computational results for the BWS case are also reported.

The literature on the blocking job shop scheduling problem includes several recent algorithmic contributions specially tailored for the BWS version of the problem. Gröflin and Klinkert (2009) propose a tabu search algorithm in which the solution feasibility is recovered after every move. In Gröflin et al. (2011), the neighborhood is enlarged to take into account machine flexibility and improved results are presented. A complete local search with memory is proposed by Zhu and Li (2011) which compares favourably with the greedy heuristic algorithms presented in Mascis and Pacciarelli (2002). Recently, Oddi et al. (2012) presented an iterative flattening search for the BWS problem obtaining state-of-the-art results.

The literature for the BNS case experienced a smaller number of contributions. Mati et al. (2001a, b) study a flexible manufacturing system and model the production scheduling problem as a BNS job shop scheduling problem in which each operation may require more than a resource at a time to be executed. They develop a geometric approach to optimally solve the BNS problem with two jobs. Mati and Xie (2011) extend the previous works to include resource flexibility. They propose an insertion heuristic based on the geometric approach for the problem with two jobs and incorporate this algorithm in a tabu search framework to produce deadlock-free schedules for the problem with n jobs.

In this paper, we discuss and apply a conceptually very simple technique, known as *Iterated Greedy* (IG), which aims at improving the performance of a greedy constructive heuristic procedure in short computation times.

The paper is organized as follows. In Sect. 2 we formulate job shop scheduling problems with blocking constraints using the alternative graph formulation. In Sect. 3 we present our iterated greedy algorithm for solving the problem modeled by means of the alternative graph formulation. Computational results on a set of benchmark instances are presented in Sect. 4, and finally some conclusions follow.

2 The job shop scheduling problem with blocking constraints

In the job shop scheduling problem a set of jobs J must be processed on a set of machines M, each processing at most one job at a time. The processing of a job on a machine is called an *operation* and cannot be interrupted. We let $\{o_1, \dots, o_n\}$ be the set of all operations. The sequence of operations for each job is prescribed, while the sequence of operations for each machine has to be determined in such a way that the time needed to complete all operations, called the *makespan*, is minimum. More formally, the scheduling problem consists in assigning a *starting time* t_i to operation $o_i, i = 1, \ldots, n$, such that: (i) precedence constraints between consecutive operations of the same job are satisfied; (ii) each machine hosts at most one job at a time; and (iii) the makespan is minimized.

In the blocking job shop scheduling problem no intermediate storage is allowed between two consecutive machines. Hence, once a job completes processing on machine M_h it either moves to the subsequent machine M_k (if it is available) or it remains on M_h, thus blocking it (if M_k is not available). The blocking job shop scheduling problem can be modeled by using the alternative graph model of Mascis and Pacciarelli (2002). An alternative graph is a triple $\mathcal{G} = (N, F, A)$. $N = \{0, 1, \ldots, n+1\}$ is a set of nodes, and each node $i \in N$ is associated to operation o_i. Two additional nodes start 0 and finish $n+1$ (dummy operations o_0 and o_{n+1}) model the beginning and the completion of the schedule, i.e., o_0 precedes all other operations while o_{n+1} follows all other operations. F is a set of *fixed* directed arcs and f_{ij} is the length of the fixed arc $(i, j) \in F$. A fixed arc (i, j) represents a precedence constraint on the starting time of operation o_j, which is constrained to start at least f_{ij} time units after o_i, i.e., $t_j \geq t_i + f_{ij}$. Finally, A is a set of m pairs of *alternative* directed arcs, and a_{ij} is the length of the alternative arc (i, j). If $((i, k), (h, j)) \in A$, then arcs (i, k) and (h, j) form an *alternative pair*. Alternative pairs are used to model precedence relations between pairs of operations to be processed on the same machine. Let o_i and o_j be two operations to be processed on the same machine, and let o_h and o_k be the operations immediately following o_i and o_j in their respective jobs (see Fig. 1). The blocking constraint states that if o_i is processed before o_j, operation o_j can start the processing only after the starting time of o_h (i.e., when o_i leaves the machine). Hence, we represent this situation with the alternative arc (h, j). Conversely, arc (k, i) represents operation o_j preceding o_i.

A *selection* S is a set of arcs obtained from A by choosing at most one arc from each pair, and $\mathcal{G}(S)$ indicates the graph $(N, F \cup S)$. The selection is feasible if there are no positive length cycles in the resulting graph $\mathcal{G}(S)$. In fact, a positive length cycle represents an operation preceding itself, which is not feasible. A selected alternative arc (i, j) represents the precedence relation $t_j \geq t_i + a_{ij}$.

When swaps are allowed (Fig. 1a) the length of all the alternative arcs is set to zero. In fact given an alternative pair $((i, k), (h, j)) \in A$, o_j can start immediately after the start of o_h, i.e., $a_{hj} = 0$. Note that, in this case, a cycle of alternative arcs has total length zero, and it is therefore feasible. In the no-swap case a deadlock is infeasible. Since a deadlock corresponds to a cycle of alternative arcs (see Fig. 2), this infeasibility is achieved by setting the length of alternative arcs to a small positive value ε (Fig. 1b). Positive length models the fact that an operation can start processing on a machine strictly after that the previous operations has left it.

Given a feasible selection S, we denote the length of a longest path from i to j in $\mathcal{G}(S)$ by $l^S(i, j)$ (if there is no path from i to j we assume $l^S(i, j) = -\infty$). A *solution* S is a selection of $|A|$ alternative precedence relations, exactly one from each pair in A. A feasible solution is *optimal* if the length $l^S(0, n+1)$ of a longest path from 0 to $n+1$ is minimum over all feasible solutions.

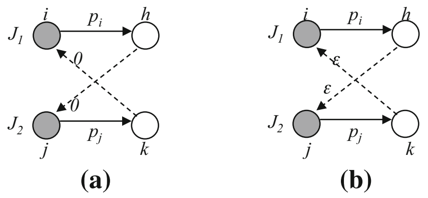

Fig. 1 **a** The alternative pair of blocking operations with swaps allowed. **b** The alternative pair of blocking operations when swaps are not allowed

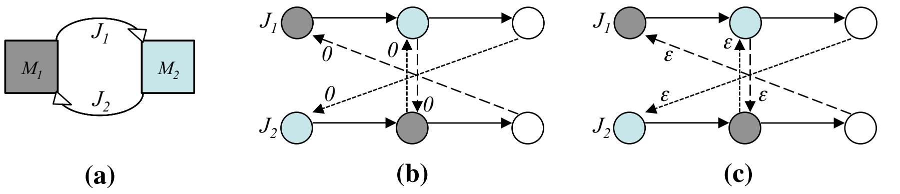

Fig. 2 **a** A simple swap cycle between two operations. **b** The alternative graph model with swaps allowed. **c** The alternative graph model when swaps are not allowed

Finally, given a feasible selection S, we define L^S_{ij} as the length of a critical (i.e., longest) path passing through the alternative arc (i, j):

$$L^S_{ij} = l^S(0, i) + a_{ij} + l^S(j, n + 1).$$

L^S_{ij} represents a rough lower bound of the longest path between 0 and $n+1$ (i.e., of the *makespan*) in any feasible solution obtained starting from the selection $S \cup \{(i, j)\}$.

We also make use of *static implications*. Two alternative pairs $((k, i), (h, j))$ $((d, a), (c, b))$ are statically implied if there is no feasible solution in which arcs (k, i) and (c, b) are selected or arcs (h, j) and (d, a) are selected. In other words, the selection of arc (d, a) implies the selection of arc (k, i), and the selection of arc (h, j) implies the selection of arc (c, b). Given an alternative arc (h, i), we define the set $I(h, j)$ as the set of alternative arcs statically implied by (h, j). Whenever an alternative arc pair (h, j) is selected/deselected also all arcs in $I(h, i)$ are selected/deselected.

The following simple result allows to establish a correspondence between the selection of arcs from different alternative pairs.

Proposition 1 (Implication) *Given a selection S and two unselected alternative pairs $((d, a), (c, b))$ and $((h, j), (k, i))$, if $l^S(a, h) \geq 0$ and $l^S(j, d) \geq 0$ then there is no feasible solution in which arcs (d, a) and (h, j) are both selected in S.*

Proposition 1 can be applied in particular with the empty selection $S = \emptyset$, when the graph $\mathcal{G}(\emptyset) = (\mathcal{N}, \mathcal{F})$ is composed of a set of chains of fixed arcs. Then, the paths $l^S(a, h) \geq 0$ and $l^S(j, d) \geq 0$ can be composed only by fixed arcs, i.e., the paths exist only if nodes a, h refer to a same job J_1, and j, d refer to another job J_2, sharing some machines with J_1. We call this implication static since it holds for any selection S. The conditions of Proposition 1 hold for $S = \emptyset$ in the following two cases:

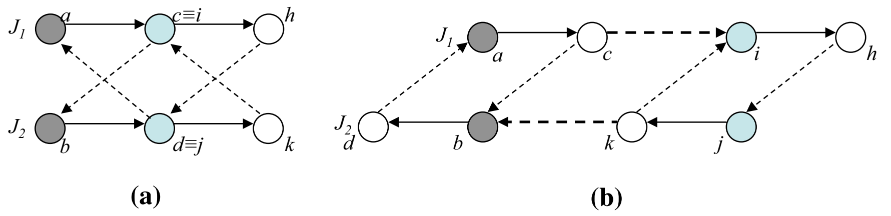

Fig. 3 **a** Jobs passing in the same order. **b** Jobs passing in opposite order

```
Iterated Greedy
Input:
    Initial solution $S$
begin
    while stopping criterion is not fulfilled do
        $S'$ = Destruction( $S$ )
        $S''$ = Construction( $S'$ )
        $S$ = Acceptance Criterion( $S$, $S''$)
    end
end
```

Fig. 4 The pseudocode of an iterated greedy algorithm

- J_1 and J_2 pass both through two consecutive machines M_1 and M_2 in the same order. In this case $c \equiv i$ and $d \equiv j$ (see Fig. 3a).
- J_1 and J_2 pass both through M_1 and M_2 consecutively, although in the opposite order (see Fig. 3b).

Whenever one of the two above situations occurs, the selection of arc (d, a) implies the selection of arc (k, i), and the selection of arc (h, j) implies the selection of arc (c, b). Therefore $I(d, a) = \{(k, i)\}$, $I(k, i) = \{(d, a)\}$, $I(h, j) = \{(c, b)\}$ and $I(c, b) = \{(h, j)\}$.

Static implications can be computed off-line very efficiently and have been successfully applied to train scheduling problems (D'Ariano et al. 2007).

3 Iterated greedy

Iterated Greedy is a conceptually very simple approach that can be used to improve the performance of a greedy heuristic, which is used as a black box within the algorithm. The basic idea is to iterate a Destruction–Construction cycle. A sketch of the algorithm is given in Fig. 4. A Destruction–Construction cycle consists of three different phases, namely *destruction* (some components of the incumbent solution are removed), *construction* (starting from the partial solution a new complete solution is built by the greedy heuristic) and *acceptance criterion* (the new solution is evaluated and possibly accepted as a new incumbent). Finally a *stopping criterion* is adopted to terminate the search.

Given its simple structure, the IG algorithm has been independently developed by different research communities under different names, such as Ruin and Recreate (2000), Adaptive Large Neighborhood Search (Shaw 1997) or Iterated Flattening Search (Cesta et al. 2000). IG algorithms have been successfully applied to the Set Covering Problem (Marchiori and Steenbeek 2000), to some variants of the Vehicle Routing Problem (Ropke and Pisinger 2006a, b), to the permutation flowshop scheduling in both versions without and with blocking (Ruiz and Stützle 2007, 2008; Ribas et al. 2011) and to the BJSS problem (Oddi et al. 2012).

One of the main advantages of the iterated greedy algorithm is that it does not rely on specific properties of the problem addressed, but all the problem specific knowledge is embedded in the underlying heuristic solution procedure. This is particularly useful when dealing with very general problems that suffer from a lack of specific properties to be used in the design of efficient solution algorithms. Moreover, the framework is easy to implement once a construction heuristic procedure is available. In the remaining part of this section we describe the three phases of our iterated greedy algorithm for the blocking job shop problem with or without swaps.

3.1 Initial solution

The first incumbent solution is obtained by simply ordering the jobs from 1 to $|J|$ and then, for each alternative pair involving jobs J_i and job J_j with $i < j$, selecting the arc in which J_i precedes J_j. Is it easy to show that the initial solution constructed in this way is always feasible in the BWS and BNS cases, since a cycle in $\mathcal{G}(S)$ would require at least one arc from J_j to J_i, with $i < j$. This is relevant since, starting from a partial selection, the problem of deciding whether a feasible solution exists is a NP-complete problem for the BNS case (Mascis and Pacciarelli 2002). However we observe as the solution found by this initial heuristic has typically very poor quality.

3.2 Destruction

In the Destruction phase a portion of the current selection S (representing the incumbent solution) is deselected, and a new partial selection S' is obtained. More precisely, each alternative pair is randomly deselected with probability p. The probability p is a parameter of the Destruction procedure called *perturbation strength* and in our test we set it to 0.8, i.e., in the destruction phase 80 % of all alternative arcs are deselected. Recall that, since we use static implications, every time the algorithm deselects the alternative arc $(i, j) \in ((i, j), (h, k))$ all the implied alternative arcs in $I(i, j)$ are also deselected. Thus the actual fraction of removed arcs is at least p.

Notice that, in our model, k operations that must be processed on the same machine lead to $k(k-1)/2$ alternative pairs. Clearly, only $(k-1)$ of them are necessary, while all the others are redundant. Removing a redundant arc would not lead to a different solution, since a redundant arc is implied by the others and construction algorithm would select it immediately. Hence, in order to reach a new solution, the destruction phase must remove a large number of arcs, most of which are redundant.

```
Constructive Heuristic
Input:
    Alternative graph $\mathcal{G} = (N, F, A)$
    Initial Selection $S$
    if AMCC then criterion $\mathcal{C} = \max\{L^S_{ij}, L^S_{hk}\}$ is maximum
    if SMBP then criterion $\mathcal{C} = |L^S_{ij} - L^S_{hk}|$ is minimum
    if AMSP then criterion $\mathcal{C} = L^S_{ij} + L^S_{hk}$ is maximum
begin
    while $A \neq \emptyset$ do
        Let $\mathcal{G} = (N, F \cup S)$
        Choose pair $((i, j), (h, k)) \in A$ according to criterion $\mathcal{C}$ and let $L^S_{ij} \leq L^S_{hk}$.
        Let $S_1 = S \cup \{(i, j)\} \cup I(i, j)$.
        if $\mathcal{G}(S_1)$ is infeasible then let $S_2 = S \cup \{(h, k)\} \cup I(h, k)$.
            if $\mathcal{G}(S_2)$ is infeasible then STOP, failed in finding a feasible solution.
            else Update $S = S_2$ and remove from $A$ all pairs $(h, k)$ and $I(h, k)$ belong to.
        else Update $S = S_1$ and remove from $A$ all pairs $(i, j)$ and $I(i, j)$ belong to.
    end
end.
```

Fig. 5 Sketch of the constructive algorithm

3.3 Construction

Given a partial selection S, a constructive greedy procedure is applied to extend S at each step with a new alternative arc (i, j), i.e., $S = S \cup \{(i, j)\} \cup I(i, j)$, until a new feasible solution S' is obtained or an infeasibility is detected. The latter case arises when there is an alternative pair $((i, j), (h, k))$ such that both $\mathcal{G}(S \cup \{(i, j)\} \cup I(i, j))$ and $\mathcal{G}(S \cup \{(h, k)\} \cup I(h, k))$ contain a positive cycle, and we say that S is infeasible even if $\mathcal{G}(S')$ does not contain any cycle.

As constructive heuristic we next describe a family of greedy heuristics described in Fig. 5, differing from each other for a selection criterion $\mathcal{C}$. We investigate three criteria, named AMCC (Avoid Maximum Current Completion time), AMSP (Avoid Max Sum Pair) and SMBP (Select Most Balanced Pair). The first is introduced in Mascis and Pacciarelli (2002) while the others are introduced in Pranzo et al. (2003). The rationale behind this choice is that moving from the former to the latter the probability of finding feasible schedules increase at the price of a deteriorating makespan. The AMCC heuristic chooses at each step the alternative pair $((i, j), (h, k)) \in A$ such that $\max\{L^S_{ij}, L^S_{hk}\}$ is maximum over all unselected alternative pairs. In other words, AMCC avoids the selection of the arc that would increase most the completion time of the current partial schedule. This choice has been demonstrated to be quite effective with different versions of the job shop scheduling problem (Laborie 2003; Gabel and Riedmiller 2007; D'Ariano et al. 2007). The AMSP heuristic chooses at each step the alternative pair $((i, j), (h, k)) \in A$ such that $L^S_{ij} + L^S_{hk}$ is maximum over all unselected alternative pairs, while the SMBP heuristic chooses at each step the alternative pair $((i, j), (h, k)) \in A$ such that $|L^S_{ij} - L^S_{hk}|$ is minimum. Once the pair is chosen, in all cases the heuristic selects the most promising arc of the pair, i.e., if $L^S_{ij} < L^S_{hk}$, then $(i, j) \cup I(i, j)$ is selected.

If the selection of arc (i, j) would cause a positive length cycle in the graph, then the arc (h, k) is selected. If also the selection of (h, k) would lead to a positive length cycle the algorithm terminates and returns the (acyclic but infeasible) partial selection.

Observe that, as stand-alone heuristic, AMCC often fails in finding a feasible solution for large instances of both the BWS and BNS case. Hence, the use of a metaheuristic scheme is particularly useful to achieve the feasibility of the solutions, besides improving their quality.

3.4 Acceptance criterion

Once a candidate solution is built, the algorithm decides whether to accept it as the new incumbent or to discard it and retain the previous incumbent. Among the acceptance criteria defined in the literature, in our experiments we assessed the following two:

- Random Walk (RW). When RW is adopted every feasible candidate solution generated by the constructive phase is always accepted. If the construction phase does not terminate with a feasible solution then the IG maintains the previous incumbent solution.
- Simulated Annealing like (SA). In the Simulated Annealing criterion (Ruiz and Stützle 2007; Ropke and Pisinger 2006b) a new feasible solution S' is always accepted if it is not worst than the incumbent S, i.e., if $C_{max}(S') \leq C_{max}(S)$. If $C_{max}(S') > C_{max}(S)$, then the candidate solution S' is accepted with probability $e^{[C_{max}(S)-C_{max}(S')]/T}$ where T is a parameter called the temperature. In our experiments we set $T = 0.5$.

3.5 Stopping criterion

Different stopping criteria can be devised, such as a time limit, a maximum number of iterations and so on. In our tests we set a maximum time limit of computation of 60 s for each run. To avoid stagnation and to improve the performance of the algorithm we run the IG for 10 independent runs and retain the best solution found.

4 Computational results

We tested the performance of our algorithm on the 58 benchmark instances for the job shop scheduling problem by Fisher and Thompson (1963), Lawrence (1984), Applegate and Cook (1991), Adams et al. (1988), adapted to the BWS and BNS cases (for a total of 116 instances). The optimum for these instances is known only for the 18 BWS and 18 BNS instances with 10 jobs and 10 machines (Mascis and Pacciarelli 2002). The time needed to prove the optimality ranges from few minutes up to 30 hours on a Pentium II at 350 MHz. In what follows we refer to this algorithm as BB.

In this section, we first briefly discuss the preliminary test phase. Since the results obtained in this phase often improve the best known solutions from the literature, we show the updated best known upper bound for each instance. Then we compare the performance of the two selected IG configurations against the best performing algorithms for both the BWS (Sect. 4.1) and BNS (Sect. 4.2) problems.

In a preliminary test phase, we tested 12 different parameters setting configurations, retaining the best solution found for all the runs. More specifically we considered three perturbation strength {0.4, 0.6, 0.8} two acceptance criteria (Random Walk and Simulated Annealing) and two different construction phases (running the AMCC or taking the best among three independent runs using AMCC, AMSP and SMBP criteria.

From this preliminary test phase we selected the two configurations exhibiting the best performance:

– *Destruction* In the destruction phase we set perturbation strength equal to 0.8.
– *Construction* The chosen constructive greedy heuristic is the AMCC algorithm.
– *Acceptance Criterion* The two best configurations differ for the acceptance criterion, which is Random Walk (RW) in one configuration and Simulated Annealing (SA) in the other.
– *Stopping Criterion* As stopping criterion we use a time limit of computation equal to 10 min. After every 60 s the current run is interrupted and a new independent run is started.

We refer to these algorithms as IG.RW and IG.SA, respectively. The algorithm is implemented in C++, uses a single thread and runs on an 3.0 GHz Core Duo Intel processor equipped with 2 Gb of RAM.

In the Appendix of the paper (Table 5) we report on the number of iterations executed by IG in a second of computation. This value varies from more than one thousand for the smallest (10×5) instances to less than ten for the largest (30×10) instances. To give an idea of the dynamic behavior of IG during its execution, for the two instances ft10 (with and without swap allowed) on average after 10,000 moves, the percentage of failures in the construction phase (when AMCC fails in finding a feasible solution) is about 31 % for IG.SA and 44 % for IG.RW, the improving moves (new solution feasible and better than the previous incumbent) are 8 % for IG.SA and 14 % for IG.RW, the side-ways moves (the new incumbent has the same makespan than the previous one) are more than 44 % for IG.SA and about 32 % for IG.RW, the worsening moves are about 10 % for IG.RW and 18 % for IG.SA, 8 % of which of the latter are accepted.

4.1 Performance analysis on blocking with swap

In this section, we compare our results with the best known solutions available in the literature. As a benchmark for the BWS case, we consider two tabu search algorithms (Gröflin and Klinkert 2009; Gröflin et al. 2011), a rollout algorithm (Meloni et al. 2004), a complete local search with memory (Zhu and Li 2011), simple heuristic techniques reported in Pranzo et al. (2003), Liu and Kozan (2011) and the recent algorithms presented in Oddi et al. (2012). The best performing algorithms described in these papers are the tabu search algorithm by Gröflin and Klinkert (2009) and Gröflin et al. (2011) (denoted as GK and GPB respectively), the iterated flattening scheme algorithm (Oddi et al. 2012) denoted as IFS and the CP-OPT constraint programming solver of IBM Ilog (Oddi et al. 2012) which implements the approach described in

Laborie and Godard (2007). Note that the two latter algorithms have the same structure of the IG, as they both repeat a destruction–construction cycle, although with different components. Specifically, in Oddi et al. (2012) two destruction criteria are analyzed, namely a random selection of the relaxed activities and a slack-based destruction procedure, which restricts the pool of relaxable activities to the subset containing those activities that are closer to the critical path. The construction component has the same structure of our constructive heuristic, but the criterion $\mathcal{C}$ is different from the ones used by AMCC, SMBP, AMSP. Finally, a solution is then accepted only if it improves the current best solution. Laborie and Godard (2007) use a portfolio of destruction and construction components. At each destruction–construction iteration, a learning algorithm chooses the specific component to use by taking into account the quality of the improvements and the computation times attained in the previous iterations.

In Table 1 we compare the results found by our IG against the best known solutions from the literature. The first two columns report the instance name and size (number of jobs $\times$ number of machines). The reference to the paper and the best known solution for each instance are shown in Column 3 and 4, respectively. Finally, in Column 5 we show the best result found by all the tested configurations of IG, highlighted in bold when some configuration of the IG improves or attains the best known solution.

The overall CPU time required for each instance by the 12 configurations of IG to obtain the results in Table 1 is 7,200 s on a 3.0 GHz processor. The CPU times required by the other best performing algorithms are as follows. GK performs 5 independent runs of 1,800 s each on a 2.8 GHz processor, for a total of 9,000 s. IFS requires two runs of 1,800 s each. Since 16 configurations of IFS are presented in Oddi et al. (2012) and Table 1 shows the best result, the total computation time required by IFS for one instance is 16 hours of computation. CP-OPT performs a single run of 1,800 s. Both the IFS and CP-OPT run on a AMD Phenom II X4 Quad 3.5 GHz.

The 12 configurations of IG presented in this paper are able to attain or to improve the best solution known in the literature in 42 over 58 benchmark instances. Over the 40 instances for which a proven optimum is not known, IG finds a new best known in 19 cases. Among the other 18 instances for which the optimum is given in Mascis and Pacciarelli (2002), IG achieves the optimum in 17 cases.

Table 2 compares the results of the two best configurations of IG with the three best performing algorithms from the literature: GPB, IFS and CP-OPT. We restrict the comparison to the 40 BWS instances of the Lawrence benchmark, the only instances for which a published result is available for all the three benchmark algorithms. Under IFS we report the results of the best performing configuration among the 16 proposed variants presented in Oddi et al. (2012), namely the one using the slack-based procedure and with $\gamma = 0.5$.

In Table 2, columns from 1 to 4 show the instance name, size, the value of the best known solution from Table 1 and the paper in which the solution has been presented, respectively. Columns 5, 6 and 7 report the results obtained by GBP, IFS and CP-OPT, respectively. Finally, Columns 8 and 9 report the results achieved by two Iterated Greedy configurations IG.RW and IG.SA. The bold values in the table show the IG configuration attaining the new best known result. For each algorithm we present, in

Table 1 Comparisons between the best known solutions for BWS from the literature and the best solutions found by IG

Instance	$\|J\| \times \|M\|$	Reference	Best known	Best IG	Instance	$\|J\| \times \|M\|$	Reference	Best known	Best IG
abz5	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,468**	**1,468**	abz9	(20 × 15)	GK—Gröflin and Klinkert (2009)	1,166	**1,111**
abz6	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,145**	**1,145**	ft06	(6 × 6)	GK—Gröflin and Klinkert (2009)	**63**	**63**
abz7	(20 × 15)	GK—Gröflin and Klinkert (2009)	1,224	**1,106**	ft10	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,068**	**1068**
abz8	(20 × 15)	GK—Gröflin and Klinkert (2009)	1,226	**1,099**	ft20	(20 × 5)	GK—Gröflin and Klinkert (2009)	2,047	**1434**
la01	(10 × 5)	CP-OPT - IFS—Oddi et al. (2012)	**793**	**793**	la21	(15 × 10)	IFS—Oddi et al. (2012)	1,521	**1,490**
la02	(10 × 5)	IFS—Oddi et al. (2012)	**793**	**793**	la22	(15 × 10)	CP-OPT—Oddi et al. (2012)	1,379	**1,339**
la03	(10 × 5)	IFS—Oddi et al. (2012)	**715**	**715**	la23	(15 × 10)	CP-OPT—Oddi et al. (2012)	1,497	**1,445**
la04	(10 × 5)	IFS—Oddi et al. (2012)	**743**	**743**	la24	(15 × 10)	IFS—Oddi et al. (2012)	1,498	**1,434**
la05	(10 × 5)	CP-OPT - IFS—Oddi et al. (2012)	**664**	**664**	la25	(15 × 10)	IFS—Oddi et al. (2012)	1,424	**1,392**
la06	(15 × 5)	IFS—Oddi et al. (2012)	**1,064**	1,102	la26	(20 × 10)	CP-OPT—Oddi et al. (2012)	2,035	**1,989**
la07	(15 × 5)	IFS—Oddi et al. (2012)	1,038	**1,020**	la27	(20 × 10)	IFS—Oddi et al. (2012)	2,104	**2,017**
la08	(15 × 5)	IFS—Oddi et al. (2012)	**1,062**	1,071	la28	(20 × 10)	IFS—Oddi et al. (2012)	**2,027**	2,039
la09	(15 × 5)	IFS—Oddi et al. (2012)	1,185	**1,162**	la29	(20 × 10)	CP-OPT—Oddi et al. (2012)	1,898	**1,846**
la10	(15 × 5)	IFS—Oddi et al. (2012)	**1,110**	1,134	la30	(20 × 10)	IFS—Oddi et al. (2012)	2,095	**2,049**
la11	(20 × 5)	IFS—Oddi et al. (2012)	**1,466**	1,498	la31	(30 × 10)	CP-OPT—Oddi et al. (2012)	**2,921**	3,018

Table 1 continued

Instance	$\lvert J\rvert \times \lvert M\rvert$	Reference	Best known	Best IG	Instance	$\lvert J\rvert \times \lvert M\rvert$	Reference	Best known	Best IG
la12	(20×5)	IFS—Oddi et al. (2012)	1,272	**1,271**	la32	(30×10)	CP-OPT—Oddi et al. (2012)	**3,237**	3,338
la13	(20×5)	IFS—Oddi et al. (2012)	**1,465**	1,482	la33	(30×10)	CP-OPT—Oddi et al. (2012)	**2,844**	2,909
la14	(20×5)	CP-OPT—Oddi et al. (2012)	**1,506**	1,513	la34	(30×10)	CP-OPT—Oddi et al. (2012)	**2,848**	3,016
la15	(20×5)	IFS—Oddi et al. (2012)	**1,517**	**1,517**	la35	(30×10)	CP-OPT—Oddi et al. (2012)	**2,923**	3,133
la16	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,060**	**1,060**	la36	(15×15)	IFS—Oddi et al. (2012)	1,793	**1,755**
la17	(10×10)	BB—Mascis and Pacciarelli (2002)	**929**	**929**	la37	(15×15)	CP-OPT—Oddi et al. (2012)	1,952	**1,870**
la18	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,025**	**1,025**	la38	(15×15)	IFS—Oddi et al. (2012)	**1,708**	1,728
la19	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,043**	**1,043**	la39	(15×15)	IFS—Oddi et al. (2012)	1,783	**1,731**
la20	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,060**	**1,060**	la40	(15×15)	IFS—Oddi et al. (2012)	1,777	**1,743**
orb01	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,175**	**1,175**	orb06	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,199**	**1,199**
orb02	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,041**	**1,041**	orb07	(10×10)	BB—Mascis and Pacciarelli (2002)	**483**	492
orb03	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,160**	**1,160**	orb08	(10×10)	BB—Mascis and Pacciarelli (2002)	**995**	**995**
orb04	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,146**	**1,146**	orb09	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,039**	**1,039**
orb05	(10×10)	BB—Mascis and Pacciarelli (2002)	**995**	**995**	orb10	(10×10)	BB—Mascis and Pacciarelli (2002)	**1,146**	**1,146**

Table 2 Performance of IG.RW and IG.SA compared to the best known solutions (Column 3) and the best performing algorithms [GPB Gröflin et al. (2011), CP-OPT and IFS Oddi et al. (2012)] for BWS on Lawrence instances

Instance	$\|J\| \times \|M\|$	Best known	Reference	GPB	CP-OPT	IFS	IG.RW	IG.SA
la01	(10 × 5)	793	CP-OPT - IFS	820	793	793	793	793
la02	(10 × 5)	793	IFS	817	815	793	793	793
la03	(10 × 5)	715	IFS	740	790	740	715	715
la04	(10 × 5)	743	IFS	764	784	776	743	743
la05	(10 × 5)	664	CP-OPT - IFS	666	664	664	664	664
la06	(15 × 5)	1,064	IFS	1,180	1,131	1,112	1,102	1,120
la07	(15 × 5)	1,038	IFS	1,084	1,106	1,081	1,062	1,040
la08	(15 × 5)	1,062	IFS	1,125	1,129	1,135	1,089	1,093
la09	(15 × 5)	1,185	IFS	1,223	1,267	1,257	1,192	1,192
la10	(15 × 5)	1,110	IFS	1,203	1,168	1,158	1,140	1,151
la11	(20 × 5)	1,466	IFS	1,584	1,520	1,501	1,550	1,529
la12	(20 × 5)	1,272	IFS	1,391	1,308	1,321	1,342	1,332
la13	(20 × 5)	1,465	IFS	1,541	1,528	1,471	1,531	1,523
la14	(20 × 5)	1,506	CP-OPT	1,620	1,506	1,567	1,538	1,545
la15	(20 × 5)	1,517	IFS	1,630	1,571	1,547	1,593	1,575
la16	(10 × 10)	1,060	BB	1,142	1,150	1,086	1,060	1,060
la17	(10 × 10)	929	BB	977	996	1,000	930	930
la18	(10 × 10)	1,025	BB	1,078	1,135	1,120	1,040	1,061
la19	(10 × 10)	1,043	BB	1,093	1,108	1,077	1,043	1,068
la20	(10 × 10)	1,060	BB	1,154	1,119	1,166	1,080	1,080
la21	(15 × 10)	1,521	IFS	1,545	1,579	1,521	1,514	1,490
la22	(15 × 10)	1,379	CP-OPT	1,458	1,379	1,490	1,368	1,390
la23	(15 × 10)	1,497	CP-OPT	1,570	1,497	1,538	**1,445**	1,473
la24	(15 × 10)	1,498	IFS	1,546	1,523	1,498	**1,434**	1,498
la25	(15 × 10)	1,424	IFS	1,499	1,561	1,424	**1,422**	1,473
la26	(20 × 10)	2,035	CP-OPT	2,125	2,035	2,179	2,013	**2,012**
la27	(20 × 10)	2,104	IFS	2,175	2,155	2,172	2,044	**2,017**
la28	(20 × 10)	2,027	IFS	2,071	2,062	2,132	2,039	2,155
la29	(20 × 10)	1,898	CP-OPT	1,990	1,898	1,963	1,928	**1,860**
la30	(20 × 10)	2,095	IFS	2,097	2,147	2,125	2,137	**2,085**
la31	(30 × 10)	2,921	CP-OPT	3,137	2,921	3,771	3,095	3,115
la32	(30 × 10)	3,237	CP-OPT	3,316	3,237	3,852	3,415	3,420
la33	(30 × 10)	2,844	CP-OPT	3,061	2,844	3,741	2,970	2,958
la34	(30 × 10)	2,848	CP-OPT	3,146	2,848	3,796	3,016	3,057
la35	(30 × 10)	2,923	CP-OPT	3,171	2,923	3,818	3,193	3,169
la36	(15 × 15)	1,793	IFS	1,919	1,952	1,891	**1755**	1764
la37	(15 × 15)	1,952	CP-OPT	2,029	1,952	1,983	**1,870**	1,884
la38	(15 × 15)	1,708	IFS	1,828	1,880	1,708	1,728	1,720
la39	(15 × 15)	1,783	IFS	1,882	1,813	1,848	**1,731**	1,752

Table 2 continued

Instance	$\lvert J\rvert \times \lvert M\rvert$	Best known	Reference	GPB	CP-OPT	IFS	IG.RW	IG.SA
la40	(15 × 15)	1,777	IFS	1,925	1,928	1,831	**1,743**	1,771
Error (%)		0.00		5.48	3.84	6.69	1.25	1.62

the last row of Table 2, the relative percentage error over the best known solutions ($error = 100 \times (found - best)/best$) from the literature. The computation times of the algorithms are as follows: GPB uses 5 independent runs of 1,800 s each on a 3.0 GHz processor. IFS'12 requires two runs of 1,800 s each on a 3.5 GHz processor. CP-OPT performs a single run of 1,800 s on a 3.5 GHz processor. Finally, the computation time of the IG.RW and IG.SA is 600 s for both algorithms, obtained with 10 independent runs of 60 s each on a 3.0 GHz processor.

From Table 2, it turns out that both IG.RW and IG.SA outperform on average the other algorithms in terms of solution quality and computation time. More in detail, when considering the aggregated performance of each algorithm over the instances of the same size (see Table 6 in the Appendix), we observe a less consistent behavior. IFS is the best performing algorithm over the (20 × 5) instances, while CP-OPT is the best performing algorithm over the (30 × 10) instances. With all the other instances both versions of IG outperform the other algorithms. In fact, both the computation time of IG.RW and its average distance from the best known is one third of the best performing algorithm from the literature (CP-OPT), i.e., 600 against 1,800 s and 1.25 % against 3.84 %, respectively. Moreover, the two IGs are able to improve the best known solution in 11 out of 40 benchmark instances (the values are shown in bold in the table).

In the Appendix of the paper (Table 7) we provide additional details on the performance of the two IG algorithms across the ten repetitions on each BWS instance (average, best, worst results and standard deviation). From Table 7, it can be noticed that even a single one-minute run of the IG yields similar average performance compared to GPB and IFS.

4.2 Performace analysis on blocking no swap

In this section we focus on the BNS case. As in the previous section, we first compare the overall performance of the 12 configurations of IG with the current state-of-the-art. We next compare the two best performing configurations against the best algorithm for JSP with blocking and no swaps. To the best of our knowledge, besides the exact approach in Mascis and Pacciarelli (2002) referred to as BB, the only available heuristic results on JSP in the BNS case are a tabu search algorithm by Mati and Xie (2011) referred to as MX, a rollout algorithm (Meloni et al. 2004) and simple heuristic techniques (Pranzo et al. 2003). Among these approaches the tabu search of Mati and Xie (MX) is the best performing algorithm. Thus we omit the other approaches in the comparison.

In Table 3 we show the current state-of-the-art performance on BNS instances of JSP. The instance name and its size are reported in the first two columns. The best

Table 3 Comparisons between the best known solutions for BNS from the literature and the best solutions found by IG

Instance	$\lvert J\rvert \times \lvert M\rvert$	Reference	Best known	Best IG	Instance	$\lvert J\rvert \times \lvert M\rvert$	Reference	Best known	Best IG
abz5	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,641**	**1,641**	abz6	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,249**	**1,249**
ft10	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,158**	**1,158**					
la01	(10 × 5)	MX—Mati and Xie (2011)	**881**	**881**	la21	(15 × 10)	MX—Mati and Xie (2011)	1,700	**1,627**
la02	(10 × 5)	MX—Mati and Xie (2011)	**900**	**900**	la22	(15 × 10)	MX—Mati and Xie (2011)	1,519	**1,426**
la03	(10 × 5)	MX—Mati and Xie (2011)	**808**	**808**	la23	(15 × 10)	MX—Mati and Xie (2011)	1,731	**1,574**
la04	(10 × 5)	MX—Mati and Xie (2011)	862	**859**	la24	(15 × 10)	MX—Mati and Xie (2011)	1,633	**1,502**
la05	(10 × 5)	MX—Mati and Xie (2011)	742	**732**	la25	(15 × 10)	MX—Mati and Xie (2011)	1,655	**1,533**
la06	(15 × 5)	MX—Mati and Xie (2011)	1,243	**1,225**	la26	(20 × 10)	MX—Mati and Xie (2011)	2,310	**2,146**
la07	(15 × 5)	MX—Mati and Xie (2011)	1,194	**1,133**	la27	(20 × 10)	MX—Mati and Xie (2011)	2,362	**2,191**
la08	(15 × 5)	MX—Mati and Xie (2011)	**1,216**	1,219	la28	(20 × 10)	MX—Mati and Xie (2011)	2,291	**2,245**
la09	(15 × 5)	MX—Mati and Xie (2011)	1,349	**1,311**	la29	(20 × 10)	MX—Mati and Xie (2011)	2,184	**2,030**
la10	(15 × 5)	MX—Mati and Xie (2011)	1,270	**1,237**	la30	(20 × 10)	MX—Mati and Xie (2011)	2,312	**2,242**
la11	(20 × 5)	MX—Mati and Xie (2011)	1,725	**1,641**	la31	(30 × 10)	MX—Mati and Xie (2011)	3,403	**3,219**
la12	(20 × 5)	MX—Mati and Xie (2011)	1,479	**1,465**	la32	(30 × 10)	MX—Mati and Xie (2011)	3,701	**3,567**
la13	(20 × 5)	MX—Mati and Xie (2011)	1,683	**1,627**	la33	(30 × 10)	MX—Mati and Xie (2011)	3,363	**3,201**
la14	(20 × 5)	MX—Mati and Xie (2011)	1,745	**1,686**	la34	(30 × 10)	MX—Mati and Xie (2011)	3,380	**3,202**
la15	(20 × 5)	MX—Mati and Xie (2011)	1,749	**1,680**	la35	(30 × 10)	MX—Mati and Xie (2011)	**3,373**	3,494
la16	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,148**	**1,148**	la36	(15 × 15)	MX—Mati and Xie (2011)	2,026	**1,835**
la17	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**968**	**968**	la37	(15 × 15)	MX—Mati and Xie (2011)	2,177	**1,931**
la18	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,077**	**1,077**	la38	(15 × 15)	MX—Mati and Xie (2011)	1,968	**1,813**
la19	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,102**	**1,102**	la39	(15 × 15)	MX—Mati and Xie (2011)	1,992	**1,811**

Table 3 continued

Instance	$\|J\| \times \|M\|$	Reference	Best known	Best IG	Instance	$\|J\| \times \|M\|$	Reference	Best known	Best IG
la20	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,118**	**1,118**	la40	(15 × 15)	MX—Mati and Xie (2011)	1,958	**1,815**
orb01	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,256**	**1,256**	orb06	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,266**	1,275
orb02	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,144**	**1,149**	orb07	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**527**	**527**
orb03	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,311**	**1,311**	orb08	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,139**	**1,139**
orb04	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,246**	**1,246**	orb09	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,130**	**1,130**
orb05	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,203**	**1,203**	orb10	(10 × 10)	BB—Mascis and Pacciarelli (2002)	**1,367**	**1,367**

known solution from the literature is shown in Column 3, while Column 4 shows the paper reporting first the best solution. Finally, in Column 5 (Best IG) we report the best solution found by the 12 IG configurations. In bold are highlighted the new best values. The proposed IG configurations are able to attain or to improve the best known solution in all but 3 instances. For the 40 instances for which a proven optimum is not known, IG finds a new best known solution in 29 cases. Moreover, for the 18 known optima, IG attains the optimum in 16 cases. Regarding the CPU times, the algorithm MX described in Mati and Xie (2011) requires a shorter CPU time since the results are obtained after 2,000 s on a 2.8 GHz processor, while the 12 IG configurations require in total 2 h on a 3 GHz processor.

In Table 4 we compare the performance obtained by the two best configurations of IG, namely IG.RW and IG.SA, against that of MX over the 40 BNS instances of the Lawrence benchmark. Columns 1 and 2 show the instance name and size,

Table 4 Performance of IG.RW and IG.SA compared to the best known solutions (Column 3)and the best performing algorithm (MX Mati and Xie (2011)) for BNS on Lawrence instances

Instance	$\lvert J\rvert \times \lvert M\rvert$	Best known	Reference	MX	IG.RW	IG.SA
la01	(10 × 5)	881	MX	881	919	881
la02	(10 × 5)	900	MX	900	906	900
la03	(10 × 5)	808	MX	808	808	808
la04	(10 × 5)	862	MX	862	**859**	**859**
la05	(10 × 5)	742	MX	742	**732**	**732**
la06	(15 × 5)	1,243	MX	1,243	1,247	1,257
la07	(15 × 5)	1,194	MX	1,194	1,173	**1,143**
la08	(15 × 5)	1,216	MX	1,216	1,224	**1,213**
la09	(15 × 5)	1,349	MX	1,349	**1,311**	1,341
la10	(15 × 5)	1,270	MX	1,270	**1,237**	1,277
la11	(20 × 5)	1,725	MX	1,725	**1,683**	1,684
la12	(20 × 5)	1,479	MX	1,479	1,525	1,524
la13	(20 × 5)	1,683	MX	1,683	1,695	**1,642**
la14	(20 × 5)	1,745	MX	1,745	1,701	**1,686**
la15	(20 × 5)	1,749	MX	1,749	1,728	**1,682**
la16	(10 × 10)	1,148	BB	1,205	1,148	1,186
la17	(10 × 10)	968	BB	1,020	968	979
la18	(10 × 10)	1,077	BB	1,156	1,077	1,099
la19	(10 × 10)	1,102	BB	1,191	1,124	1,124
la20	(10 × 10)	1,118	BB	1,204	1,164	1,164
la21	(15 × 10)	1,700	MX	1,700	1,653	**1,627**
la22	(15 × 10)	1,519	MX	1,519	**1,435**	**1,435**
la23	(15 × 10)	1,731	MX	1,731	**1,574**	1,628
la24	(15 × 10)	1,633	MX	1,633	1,584	**1,530**
la25	(15 × 10)	1,655	MX	1,655	**1,558**	1,569
la26	(20 × 10)	2,310	MX	2,310	**2,159**	2,164

Table 4 continued

Instance	$\lvert J\rvert \times \lvert M\rvert$	Best known	Reference	MX	IG.RW	IG.SA
la27	(20 × 10)	2,362	MX	2,362	2,211	**2,191**
la28	(20 × 10)	2,291	MX	2,291	2,280	**2,319**
la29	(20 × 10)	2,184	MX	2,184	2,074	**2,054**
la30	(20 × 10)	2,312	MX	2,312	**2,263**	2,342
la31	(30 × 10)	3,403	MX	3,403	3,482	3,405
la32	(30 × 10)	3,701	MX	3,701	3,649	**3,576**
la33	(30 × 10)	3,363	MX	3,363	3,326	**3,255**
la34	(30 × 10)	3,380	MX	3,380	**3,306**	3,352
la35	(30 × 10)	3,373	MX	3,373	3582	3,522
la36	(15 × 15)	2,026	MX	2,026	**1,835**	1,894
la37	(15 × 15)	2,177	MX	2,177	1974	**1,931**
la38	(15 × 15)	1,968	MX	1,968	**1,813**	1,831
la39	(15 × 15)	1,992	MX	1,992	1,820	**1,811**
la40	(15 × 15)	1,958	MX	1,958	1,818	**1,815**
Error (%)		0.00		0.84	−2.11	−2.25

respectively. The best known solutions are shown in Column 3. Except for the 5 instances with 10 jobs and 10 machines for which a proven optimum is known (Mascis and Pacciarelli 2002), the best known is given by MX. In columns 4, 5 and 6 we report the results for MX, IG.RW and IG.SA. The bold values in the table show the IG configuration attaining the new best known result. For each algorithm, the last row of Table 4 report the relative percentage error ($error = 100 \times (found - best)/best$) over the best known solution from the literature. A negative value corresponds to a performance improving on average the best known solutions. The computation times of the algorithms are as follows: MX performs 10 independent runs of 200 s on a 2.8 GHz processor, for a total of 2,000 s, while the IG is the best over 10 independent runs of 60 s on a 3.0 GHz processor, for a total of 600 s.

The results show that IG clearly outperforms MX both in terms of computation time and quality of the solutions. In particular, the two IG algorithms improve the best known solutions 28 times out of the 35 instances for which the proven optimum is not known. The best average results are attained by the IG.SA (−2.25 % on average).

In the Appendix of the paper (Table 9) we provide additional details on the performance of the two IG algorithms across the ten repetitions on each BNS instance (average, best, worst results and standard deviation). Table 6 shows that also when considering the aggregated performance of each algorithm over the instances of the same size, both versions of IG clearly outperform MX.

5 Conclusions

In this paper we developed an Iterated Greedy metaheuristic to solve the job shop scheduling problem with blocking constraints, in the two variants with and without swaps. The main advantages of the iterated greedy algorithm are the ease of implementation and the independence of the framework from specific properties of the problem addressed.

Even if the proposed approach is conceptually very simple and easy to implement, the performance obtained clearly outperforms all the previously published algorithms from the literature. A new best known solution has been found for 20 BWS instances and for 29 BNS instances. It is also important to notice that the Iterated Greedy has a broad applicability since it can be easily applied to any complex scheduling problem modeled by means of the alternative graph formulation.

Future research directions should address the study of local search based approaches for the blocking job shop scheduling problem and the adaptation of the IG algorithms to real world applications with more complex constraints.

6 Appendix

See Tables 5, 6, 7, 8 and 9.

Table 5 Speed of the proposed IG algorithms

Instance size $\|J\| \times \|M\|$	Number of alternative pairs	Speed (iterations per second)			
		BWS		BNS	
		IG.RW	IG.SA	IG.RW	IG.SA
(10 × 5)	225	1,237.52	1,221.44	1,441.32	1,458.79
(10 × 10)	450	333.43	351.38	376.52	387.83
(15 × 5)	525	307.05	314.21	361.95	362.99
(15 × 10)	1,050	81.94	85.38	92.76	95.67
(20 × 5)	950	114.07	120.45	133.45	138.83
(20 × 10)	1,900	29.74	31.49	33.41	34.81
(30 × 10)	4,350	6.92	7.59	8.82	9.01
(15 × 15)	1,575	36.11	37.58	38.87	40.33

In each row the average over the 5 Lawrence instances of the same size are reported. The instance size is expressed in terms of number of jobs and machines and in terms of number of alternative pairs in the associated alternative graph. The speed is expressed in number of iterations per second on the different instance sizes. When swaps are not allowed (BNS) the algorithm is faster because it is more difficult to obtain a feasible solution and the constructive phase is halted as soon as an unfeasibility is detected, thus speeding up the computation

Table 6 Comparisons focused on different instance sizes between GPB, CP-OPT, IFS, IG.RW and IG.SA algorithms for the BWS

$\lvert J\rvert \times \lvert M\rvert$	GPB	CP-OPT	IFS	IG.RW	IG.SA
(10 × 5)	2.61	3.76	1.59	**0.00**	**0.00**
(15 × 5)	6.57	6.26	5.19	**2.34**	2.53
(20 × 5)	7.52	2.87	**2.54**	4.57	3.88
(10 × 10)	6.35	7.65	6.52	**0.69**	1.58
(15 × 10)	4.13	3.02	2.16	−**1.83**	0.12
(20 × 10)	2.98	1.33	4.07	0.05	−**0.29**
(30 × 10)	7.28	**0.00**	28.71	6.20	6.41
(15 × 15)	6.38	5.82	2.75	−**2.00**	−1.29

The table shows the relative error compared to the best known solutions from the previous literature. In each row the average performance over the 5 Lawrence instances of the same size are reported. It can be observed that IG.RW and IG.SA attain the best results for most of the instances, while CP-OPT performs best on the largest 30 × 10 instances, though exploiting a larger computation time
The bold values highlight the best performing algorithm

Table 7 Details on the performance of the IG algorithms for the BWS case on the Lawrence testbed

Instance	$\lvert J\rvert \times \lvert M\rvert$	IG.RW				IG.SA			
		Min	Avg	Max	StdDev	Min	Avg	Max	StdDev
la01	(10 × 5)	793	795.5	815	6.52	793	821	878	24.51
la02	(10 × 5)	793	795.8	800	3.43	793	807.1	824	10.97
la03	(10 × 5)	715	715	715	0.00	715	722.4	747	9.21
la04	(10 × 5)	743	743	743	0.00	743	751.9	766	9.10
la05	(10 × 5)	664	668.1	677	4.44	664	677.5	689	6.61
la06	(15 × 5)	1,102	1,160	1,183	23.26	1,120	1,156.6	1,218	24.96
la07	(15 × 5)	1,062	1,086.5	1,113	16.55	1,040	1,073.1	1,090	14.07
la08	(15 × 5)	1,089	1,131.8	1,172	28.12	1,093	1,127	1,142	17.00
la09	(15 × 5)	1,192	1,259	1,286	25.14	1,192	1,250.9	1,292	25.91
la10	(15 × 5)	1,140	1,185.2	1,223	22.95	1,151	1,175.7	1,205	21.00
la11	(20 × 5)	1,550	1,625.1	1,699	39.37	1,529	1,575.8	1,615	27.52
la12	(20 × 5)	1,342	1,406.5	1,452	36.01	1,332	1,401.3	1,459	36.08
la13	(20 × 5)	1,531	1,602.2	1,670	38.17	1,523	1,568	1,634	36.06
la14	(20 × 5)	1,538	1,621.8	1,672	41.38	1,545	1,577.3	1,617	23.73
la15	(20 × 5)	1,593	1,680.6	1,723	36.78	1,575	1,634.6	1,681	30.04
la16	(10 × 10)	1,060	1086.1	1,118	20.34	1,060	1,121.5	1,153	24.77
la17	(10 × 10)	930	932.5	937	2.80	930	964.6	976	12.63
la18	(10 × 10)	1,040	1,056.2	1,070	10.43	1,061	1,073.4	1,122	17.84
la19	(10 × 10)	1,043	1,072.5	1,096	15.81	1,068	1,084	1,122	20.20
la20	(10 × 10)	1,080	1,086.5	1,093	5.39	1,080	1,142.2	1,174	25.90
la21	(15 × 10)	1,514	1,574.4	1,712	51.45	1,490	1,554.1	1,614	36.79
la22	(15 × 10)	1,368	1,406.3	1,476	33.43	1,390	1,424.3	1,490	32.25

Table 7 continued

Instance	$\lvert J\rvert \times \lvert M\rvert$	IG.RW				IG.SA			
		Min	Avg	Max	StdDev	Min	Avg	Max	StdDev
la23	(15 × 10)	1,445	1,491.4	1,573	33.54	1,473	1,500.3	1,561	25.93
la24	(15 × 10)	1,434	1,498.2	1,561	43.32	1,498	1,536.9	1,625	41.08
la25	(15 × 10)	1,422	1,485.6	1,583	42.31	1,473	1,523.9	1,633	40.93
la26	(20 × 10)	2,013	2,130	2,223	74.20	2,012	2,087.4	2,198	45.85
la27	(20 × 10)	2,044	2,172.8	2,290	64.04	2,017	2,109.8	2,194	55.51
la28	(20 × 10)	2,039	2,189.9	2,288	66.80	2,155	2,171.1	2,231	24.64
la29	(20 × 10)	1,928	2,017.7	2,101	53.17	1,860	1,933.7	1,992	36.92
la30	(20 × 10)	2,137	2,205.4	2,256	42.94	2,085	2,184.5	2,294	71.47
la31	(30 × 10)	3,095	3,345.3	3,521	129.55	3,115	3,209.4	3,385	87.06
la32	(30 × 10)	3,415	3,675.9	3,899	114.76	3,420	3,482.6	3643	65.60
la33	(30 × 10)	2,970	3,266.5	3,415	111.05	2,958	3,166	3,307	106.36
la34	(30 × 10)	3,016	3,321.8	3,476	131.39	3,057	3,221.4	3,338	80.87
la35	(30 × 10)	3,193	3,384.6	3,604	131.23	3,169	3,387.2	3629	127.81
la36	(15 × 15)	1,755	1,833	1,883	41.24	1,764	1,819.4	1,904	36.69
la37	(15 × 15)	1,870	1,979.9	2,063	49.36	1,884	1,954	2,033	43.24
la38	(15 × 15)	1,728	1,837.3	1,936	63.24	1,720	1,774	1,863	46.85
la39	(15 × 15)	1,731	1,811.3	1,881	44.73	1,752	1,811.4	1955	55.45
la40	(15 × 15)	1,743	1,837.2	1,975	72.01	1,771	1,792.2	1851	29.05
Error (%)		1.25	5.59	9.43		1.62	4.99	9.11	

For the two configurations we present the minimum, average, maximum and standard deviation over ten one-minute runs. When comparing the relative error of the one-minute runs with the results of GPB and IFS (Table 2) it turns out that IG is already competitive after a single one-minute run

Table 8 Comparisons focused on different instance sizes between MX, IG.RW, IG.SA algorithms for the BNS

$\lvert J\rvert \times \lvert M\rvert$	MX	IG.RW	IG.SA
(10 × 5)	0.00	0.66	**−0.34**
(15 × 5)	0.00	**−1.24**	−0.69
(20 × 5)	0.00	−0.47	**−1.80**
(10 × 10)	6.69	**1.22**	2.52
(15 × 10)	0.00	−5.25	**−5.46**
(20 × 10)	0.00	**−4.11**	−3.40
(30 × 10)	0.00	0.76	**−0.59**
(15 × 15)	0.00	**−8.48**	−8.23

The table shows the relative error compared to the best known solutions from the previous literature. In each row the average performance over the 5 Lawrence instances of the same size are reported. It can be observed that IG.SA is able to consistently improve the MX results
The bold values highlight the best performing algorithm

Table 9 Details on the performance of the IG algorithms for the BNS case on the Lawrence testbed

Instance	$\|J\| \times \|M\|$	IG.RW				IG.SA			
		Min	Avg	Max	StdDev	Min	Avg	Max	StdDev
la01	(10 × 5)	919	919	919	0.00	881	943.8	983	30.53
la02	(10 × 5)	906	910.5	924	5.95	900	915.1	941	12.58
la03	(10 × 5)	808	808	808	0.00	808	821.2	869	21.35
la04	(10 × 5)	859	859	859	0.00	859	860.1	862	1.14
la05	(10 × 5)	732	732	732	0.00	732	745.2	769	14.33
la06	(15 × 5)	1,247	1,280.4	1,318	24.96	1,257	1,303.3	1,341	23.10
la07	(15 × 5)	1,173	1,195	1,220	13.98	1,143	1,204.8	1,238	25.78
la08	(15 × 5)	1,224	1,252.6	1,283	21.06	1,213	1,274.4	1,305	25.67
la09	(15 × 5)	1,311	1,384.8	1,414	27.33	1,341	1,405.3	1,463	33.12
la10	(15 × 5)	1,237	1,325.9	1,368	32.98	1,277	1,298	1,326	16.51
la11	(20 × 5)	1,683	1,760.8	1,829	43.60	1,684	1,738.7	1,808	33.75
la12	(20 × 5)	1,525	1,591.4	1,664	42.23	1,524	1,577.6	1,626	37.20
la13	(20 × 5)	1,695	1,771.6	1,834	35.60	1,642	1,718.1	1,766	32.13
la14	(20 × 5)	1,701	1,790.9	1,875	41.54	1,686	1,754.5	1,797	30.77
la15	(20 × 5)	1,728	1,829.9	1,924	53.41	1,682	1,768.5	1,867	50.44
la16	(10 × 10)	1,148	1,179.1	1,212	22.49	1,186	1,217.4	1,255	23.58
la17	(10 × 10)	968	968	968	0.00	979	1,024.1	1,074	36.90
la18	(10 × 10)	1,077	1,108	1,122	15.14	1,099	1,178.2	1,235	35.68
la19	(10 × 10)	1,124	1,124.9	1,131	2.12	1,124	1,171.7	1,230	41.68
la20	(10 × 10)	1,164	1,168.6	1,194	9.72	1,164	1,251.3	1,285	33.67
la21	(15 × 10)	1,653	1,693.4	1,741	32.02	1,627	1,743.9	1,849	60.73
la22	(15 × 10)	1,435	1,524.7	1,613	50.53	1,435	1,528	1,593	45.39
la23	(15 × 10)	1,574	1,636.6	1,682	28.92	1,628	1,710.4	1803	51.52
la24	(15 × 10)	1,584	1,656.9	1,704	35.68	1,530	1,661.4	1,765	71.67
la25	(15 × 10)	1,558	1,602.4	1,693	44.47	1,569	1,626.9	1,715	42.82
la26	(20 × 10)	2,159	2,318.6	2,476	92.56	2,164	2,257.5	2,338	48.74
la27	(20 × 10)	2,211	2,346.7	2,471	83.82	2,191	2,320.4	2,418	62.19
la28	(20 × 10)	2,280	2,433.7	2,534	81.58	2,319	2,389.5	2,466	57.69
la29	(20 × 10)	2,074	2,143.6	2,222	51.39	2,054	2,111.6	2,167	41.20
la30	(20 × 10)	2,263	2,373.7	2,561	99.40	2,342	2,380.9	2,449	41.52
la31	(30 × 10)	3,482	3,671.8	3,884	123.38	3,405	3,523.6	3,673	86.78
la32	(30 × 10)	3,649	4,038.7	4,420	196.44	3,576	3,863.5	4,195	167.38
la33	(30 × 10)	3,326	3,601.4	3,809	170.55	3,255	3,424.8	3,656	113.60
la34	(30 × 10)	3,306	3,682.8	3,862	158.28	3,352	3,560.9	3,786	128.95
la35	(30 × 10)	3,582	4,103.7	4,537	247.47	3,522	3,879.4	4309	237.51
la36	(15 × 15)	1,835	1,936.7	2,057	60.46	1,894	1,924.3	1,999	34.39
la37	(15 × 15)	1,974	2,067.1	2,152	61.74	1,931	2,076	2,140	59.28

Table 9 continued

Instance	$\lvert J\rvert \times \lvert M\rvert$	IG.RW				IG.SA			
		Min	Avg	Max	StdDev	Min	Avg	Max	StdDev
la38	(15 × 15)	1,813	1,902.4	2,009	59.79	1,831	1,905.3	1,959	40.71
la39	(15 × 15)	1,820	1,898.4	2,006	50.78	1,811	1,876.3	1,990	51.36
la40	(15 × 15)	1,818	1,926	2,026	54.30	1,815	1,909.1	2,015	48.11
Error (%)		−2.11	2.16	6.13		−2.25	2.23	6.62	

For the two configurations we present the minimum, average, maximum and standard deviation over ten one-minute runs. When comparing the relative error of the one-minute runs with the results of MX (Table 4) it turns out that MX performs better at a price of a larger CPU time

References

Adams, J., Balas, E., Zawack, D.: The shifting bottleneck procedure for job shop scheduling. Manag. Sci. **34**(3), 391–401 (1988)

Applegate, D., Cook, W.: A computational study of the job shop scheduling problem. ORSA J. Comput. **3**(2), 149–156 (1991)

Cesta, A., Oddi, A., Smith, S.F.: Iterative flattening: a scalable method for solving multi-capacity scheduling problems. In: Proceedings of the National Conference on Artificial Intelligence, pp. 742–747 (2000)

D'Ariano, A., Pacciarelli, D., Pranzo, M.: A branch and bound algorithm for scheduling trains in a railway network. Eur. J. Oper. Res. **183**(2), 643–657 (2007)

D'Ariano, A., D'Urgolo, P., Pacciarelli, D.: Optimal sequencing of aircrafts take-off and landing at a busy airport. In: Proceedings of the 13th IEEE Conference on Intelligent Transportation Systems, pp. 1569–1574 (2010)

Fisher, H., Thompson, G.L.: Probabilistic learning combinations of local job-shop scheduling rules. In: Muth, J.F., Thompson, G.L. (eds.) Industrial Scheduling. Prentice-Hall, New Jersey, Englewood Cliffs (1963)

Gabel, T., Riedmiller, M.: On a successful application of multi-agent reinforcement learning to operations research benchmarks. In: Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL), pp. 68–75 (2007)

Grabowski, J., Pempera, J.: Sequencing of jobs in some production system. Eur. J. Oper. Res. **125**, 535–550 (2000)

Gröflin, H., Pham, D.N., Bürgy, R.: The flexible blocking job shop with transfer and set-up times. J. Comb. Optim. **22**(2), 121–144 (2011)

Gröflin, H., Klinkert, A.: A new neighborhood and tabu search for the blocking job shop. Discret. Appl. Math. **157**(17), 3643–3655 (2009)

Hall, N.G., Sriskandarajah, C.: A survey of machine scheduling problems with blocking and no-wait in process. Oper. Res. **44**, 510–525 (1996)

Laborie, P.: Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. Artif. Intell. **143**, 151–188 (2003)

Laborie, P., Godard, D.: Self-adapting large neighborhood search: application to single-mode scheduling problems. In: Proceedings MISTA-07 (2007)

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

Levner, E., Kats, V.B., Levit, V.E.: An improved algorithm for a cyclic robotic scheduling problem. Eur. J. Oper. Res. **97**, 500–508 (1997)

Liu, S.Q., Kozan, E.: Scheduling trains with priorities: a no-wait blocking parallel-machine job-shop scheduling model. Transp. Sci. **45**(2), 175–198 (2011)

Marchiori, E., Steenbeek, A.: An evolutionary algorithm for large set covering problems with applications to airline crew scheduling. Real-World Applications of Evolutionary Computing, EvoWorkshops. Lecture Notes in Computer Science, pp. 367–381 (2000)

Mascis, A., Pacciarelli, D.: Job-shop scheduling with blocking and no-wait constraints. Eur. J. Oper. Res. **143**(3), 498–517 (2002)

Mati, Y., Rezg, N., Xie, X.L.: Geometric approach and taboo search for scheduling flexible manufacturing systems. IEEE Trans. Robot. Autom. **17**(6), 805–818 (2001)

Mati, Y., Rezg, N., Xie, X.L.: A taboo search approach for deadlock-free scheduling of automated manufacturing systems. J. Intell. Manuf. **12**(5–6), 535–552 (2001)

Mati, Y., Xie, X.: Multiresource shop scheduling with resource flexibility and blocking. IEEE Trans. Autom. Sci. Eng. **8**(1), 175–189 (2011)

Meersmans, P.J.M.: Optimization of container handling systems. Ph.D. Thesis, Erasmus University Rotterdam (2002)

Meloni, C., Pacciarelli, D., Pranzo, M.: A rollout metaheuristic for job shop scheduling problems. Ann. Oper. Res. **131**(1–4), 215–235 (2004)

Oddi, A., Rasconi, R., Cesta, A., Smith, S.F.: Iterative improvement algorithms for the blocking job shop. In: Twenty-Second International Conference on Automated Planning and Scheduling (2012)

Pacciarelli, D.: Alternative graph formulation for solving complex factory-scheduling problems. Int. J. Prod. Res. **40**(15), 3641–3653 (2004)

Pacciarelli, D., Pranzo, M.: Production scheduling in a steelmaking-continuous casting plant. Comput. Chem. Eng. **28**(12), 2823–2835 (2004)

Pham, D.-N., Klinkert, A.: Surgical case scheduling as a generalized job shop scheduling problem. Eur. J. Oper. Res. **185**(3), 1011–1025 (2008)

Pranzo, M., Meloni, C., Pacciarelli, D.: A new class of greedy heuristics for job shop scheduling problems. Proceedings of the 4th International Conference on Experimental and Efficient Algorithms **2647**, 223–236 (2003)

Ribas, I., Companys, R., Tort-Martorell, X.: An iterated greedy algorithm for the flowshop scheduling problem with blocking. Omega Int. J. Manag. Sci. **39**(3), 293–301 (2011)

Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. **40**(4), 455–472 (2006)

Ropke, S., Pisinger, D.: A unified heuristic for a large class of vehicle routing problems with backhauls. Eur. J. Oper. Res. **171**(3), 750–775 (2006)

Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur. J. Oper. Res. **177**(3), 2033–2049 (2007)

Ruiz, R., Stützle, T.: An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. Eur. J. Oper. Res. **187**(3), 1143–1159 (2008)

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: Record breaking optimization results using the ruin and recreate principle. J. Comput. Phys. **159**(2), 139–171 (2000)

Shaw, P.: A new local search algorithm providing high quality solutions to vehicle routing problems. Departement of Computer Sciences, University of Strathclyde, Glasgow, Scotland, Technical Report, APES group (1997)

Zhu, J., Li, X.P.: An efficient metaheuristic for the blocking job shop problem with the makespan minimization. In: International IEEE Conference on Machine Learning and Cybernetics (ICMLC), pp. 1352–1357 (2011)