A BIASED RANDOM-KEY GENETIC ALGORITHM FOR JOB-SHOP SCHEDULING

JOSÉ FERNANDO GONÇALVES AND MAURICIO G. C. RESENDE

Abstract. This paper presents a local search, based on a new neighborhood for the job-shop scheduling problem, and its application within a biased random-key genetic algorithm. Schedules are constructed by decoding the chromosome supplied by the genetic algorithm with a procedure that generates active schedules. After an initial schedule is obtained, a local search heuristic, based on an extension of the graphical method of Akers (1956), is applied to improve the solution. The new heuristic is tested on a set of 205 standard instances taken from the job-shop scheduling literature and compared with results obtained by other approaches. The new algorithm improved the best known solution values for 57 instances.

1. Introduction

In the *job-shop scheduling problem* (JSP), we are given a set $J = \{1, \ldots, n\}$ of n jobs and a set $M = \{1, \ldots, m\}$ of m machines. Job $j \in J$ consists of n_j ordered operations $O_{j,1}, \ldots, O_{j,n_j}$, each of which must be processed on one of the m machines. Let $O = \{1, \ldots, o\}$ denote the set of all operations to be scheduled. Each operation $k \in O$ uses one of the m machines for a fixed processing time d_k. Each machine can process at most one operation at a time and once an operation initiates processing on a given machine it must complete processing on that machine without interruption. Furthermore, let P_k be the set of all the predecessor operations of operation $k \in O$. The operations are interrelated by two kinds of constraints. First, precedence constraints force each operation $k \in O$ to be scheduled after all operations in P_k are completed. Second, operation $k \in O$ can only be scheduled if the machine it requires is idle.

Let a schedule be represented by a vector of finish times $(F_1, ..., F_o)$. The job-shop scheduling problem consists in finding a feasible schedule of the operations on the machines that minimizes the makespan C_{max}, i.e., the finish time of the last operation completed in the schedule.

Not only is the JSP NP-hard, but it has been also been considered to be one of the most computationally challenging combinatorial optimization problems (Lenstra and Rinnooy Kan, 1979). Early attempts at solving the JSP considered the following approaches:

- *Exact methods*: Giffler and Thompson (1960),Brucker et al. (1994), Williamson et al. (1997), Lageweg et al. (1977), Carlier and Pinson (1989, 1990), Applegate and Cook (1991), and Sabuncuoglu and Bayiz (1999). Carlier and Pinson (1989) were the first to successfully solve the notorious 10×10 (10 jobs, 10 machines) instance of Fisher and Thompson (1963), proposed in 1963 and only solved 20 years later;
- *Heuristic procedures based on priority rules*: Giffler and Thompson (1960), French (1982), Baker and McMahon (1985) and Gray and Hoesada (1991);
- *Shifting bottleneck*: Adams et al. (1988) and Balas and Vazacopoulos (1998).

Problems of dimension 20×20 are still considered to be beyond the reach of today's exact methods. A growing number of heuristics have been proposed to find optimal or near-optimal solutions of the JSP, including:

- *Simulated annealing*: Van Laarhoven et al. (1992) and Lourenço (1995);

Date: February 2011, rev. February 2013.
Key words and phrases. Job-shop; Scheduling; Genetic algorithm; Biased random-key genetic algorithm; Heuristics; Random keys, Graphical approach.

Supported by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT - Foundation for Science and Technology, project PTDC/EGE-GES/117692/2010.
AT&T Labs Research Technical Report.

- *Tabu search*: Taillard (1994), Lourenço and Zwijnenburg (1996), Nowicki and Smutnicki (1996), Nowicki and Smutnicki (2005), Zhang et al. (2007) and Zhang et al. (2008);
- *Genetic algorithms*: Davis (1985), Storer et al. (1992), Aarts et al. (1994), Della Croce et al. (1995), Dorndorf and Pesch (1995), and Gonçalves et al. (2005);
- *GRASP*: Binato et al. (2002) and Aiex et al. (2003);
- *Other heuristics*: Lourenço (1995), Vaessens et al. (1996), Lourenço and Zwijnenburg (1996), Pardalos and Shylo (2006) and Pardalos et al. (2010).

Surveys of heuristic methods for the JSP are given in Pinson (1995), Blazewicz et al. (1996), Vaessens et al. (1996), and Cheng et al. (1996, 1999). A comprehensive survey of job-shop scheduling techniques can be found in Jain and Meeran (1999).

In this paper, we introduce a new local search neighborhood for the job-shop scheduling problem, extending the graphical method of Akers (1956) for more than two jobs. This local search is hybridized with a tabu search procedure. The hybrid local search procedure is coordinated by a biased random-key genetic algorithm (Gonçalves and Resende, 2011), or BRKGA. In computational experiments with a large set of standard job-shop scheduling test problems, we show that our algorithm is competitive with state-of-art heuristics for the JSP and improves the best known solution values for 57 of these instances.

The remainder of the paper is organized as follows. Section 2 introduces the new local search for the JSP and Section 3 describes its use within a BRKGA. This section also describes a schedule generation procedure and a solution improvement procedure. Section 4 reports experimental results. Concluding remarks are made in Section 5.

2. New local search for JSP

We present a new neighborhood for local search for the JSP based on a graphical method originally proposed by Akers (1956) for JSPs with two jobs. To illustrate the various aspects of the approach we use an instance with data shown in Table 1. This example consists of four jobs (J_1, J_2, J_3, J_4) to be processed on three machines (a, b, c).

Table 1. Problem data for 4-job, 3-machine example.

	J_1		J_2		J_3		J_4	
Seq. Order	Mach.	Proc. time	Mach.	Proc. time	Mach.	Proc. time	Mach.	Proc. time
1	a	2	b	3	c	5	b	2
2	b	3	c	2	b	2	a	4
3	c	4	a	3	a	3	c	2

In the remainder of this section, we present the original graphical approach of Akers (1956) for two jobs, and propose its extension for more than two jobs, and a new local search that makes use of the extension.

2.1. Graphical method for two jobs.

Akers (1956) introduced a graphical method for job-shop scheduling problem with two jobs. The method consists in transforming the two-job-shop scheduling problem into a shortest-path problem. This problem is represented in a two-dimensional plane with obstacles, where one axis corresponds to job $J_1 = \{O_{1,1}, O_{1,2}, \ldots, O_{1,n_1}\}$ and is decomposed into n_1 intervals and the other to job $J_2 = \{O_{2,1}, O_{2,2}, \ldots, O_{2,n_2}\}$, and is decomposed into n_2 intervals. For $i = 1, 2$ and $k = 1, \ldots, n_i$, interval $I_{i,k}$ has a length $L_{i,k,}$ that is equal to the processing time of operation $O_{i,k}$. If operations $O_{1,k}$ and $O_{2,l}$ share the same machine, then the rectangle induced by intervals $I_{1,k}$ and $I_{2,l}$ becomes an obstacle. The right and upper borders of the rectangle defined by the start point S and the end point F, correspond to the completion of the two jobs. A feasible solution of the JSP corresponds to a path that goes from point S to point F while avoiding the interiors of the obstacles. A path consists of only horizontal, vertical, and diagonal segments, where a horizontal (resp. vertical) segment implies that only J_1 (resp. J_2) is processed, whereas a diagonal segment implies that both J_1 and J_2 are processed simultaneously. The length L of a path is equal to the makespan of the corresponding schedule and is given by

$$L = L_H + L_V + \frac{L_D}{\sqrt{2}}, \tag{2.1}$$

where L_H, L_V, and L_D represent the total lengths of the horizontal, vertical, and diagonal segments, respectively. Therefore, finding the schedule that minimizes the makespan is equivalent to finding the shortest path in this plane. Figure 2.1 depicts the shortest path and the corresponding schedule for a job-shop problem consisting of jobs J_1 and J_2 defined in Table 1.

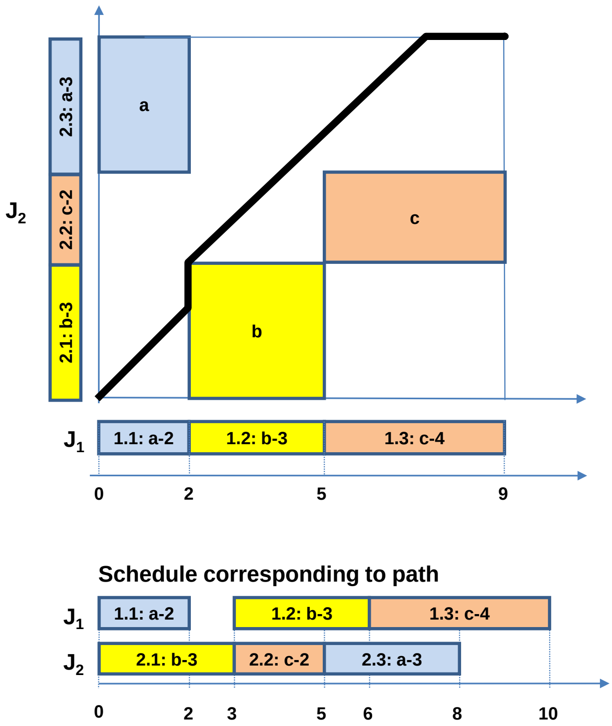

FIGURE 2.1. Akers graphical method for two jobs.

Let r denote the number of obstacles in the shortest-path problem. Brucker (1988) showed that finding the shortest path on a plane with obstacles is equivalent to finding the shortest path in an directed graph G that can be constructed in $O(r \log r)$ time and on which a shortest path can be found in $O(r)$ time, where r is bounded above by $O(n_1 n_2)$. The digraph $G = (V, E, d)$ is constructed as follows:

(1) V is the set of vertices, consisting of the start point $S = (0, 0)$, the end point F, and all the north-west (NW) and south-east (SE) corners of the obstacles;
(2) Each vertex $v \in V \setminus \{F\}$ has at most two successors, obtained by moving diagonally (at an angle of 45°) from v, until an obstacle is hit. If the obstacle encountered is the last one, then F is the unique successor of v (see Figure 2.2a). If the obstacle represents a machine conflict, then its NW and SE corners are the two direct successors of vertex v (see Figure 2.2b);
(3) When an obstacle D is hit, then two links (v, D_{NW}) and (v, D_{SE}) corresponding to the two vertices being the direct successors of vertex v are created, where D_{NW} and D_{SE} are, respectively, the NW and SE corners of obstacle D (see Figure 2.2b). The length $d(v_1, v_2)$ of link (v_1, v_2) is equal to its horizontal or vertical part plus the projection on one of the axis of its diagonal part.

A path going from S to F in digraph $G = (V, E, d)$ corresponds to a feasible schedule for the problem and its length is equal to the makespan. Therefore, finding the optimal makespan for the example is equivalent to finding a shortest path on the graph shown in Figure 2.1.

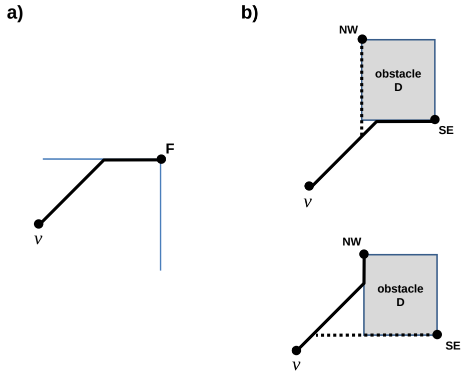

FIGURE 2.2. Successors of a vertex v.

2.2. **Extension of the graphical method for $n > 2$.** We now propose a new heuristic for solving job-shop problems with more than two jobs based on the graphical method for the two-job problem described in the previous subsection. Jobs are added to the schedule, one at a time. At each stage s, a new job is added. All jobs already scheduled are placed below the horizontal axis and the new job is placed to the left of the vertical axis. Next, the graphical method of Akers (1956) for $n = 2$ is used to find the shortest path taking into account the obstacles generated by the operations that share the same machine in the job on the vertical axis and all the jobs in the horizontal axis (see Figure 2.3a where job J_3 is added to the final schedule of jobs J_1 and J_2 in Figure 2.1). After finding the shortest path, the schedules of the job on the vertical axis and the jobs on the horizontal axis are updated accordingly (see Figure 2.3b). Finally, all jobs already scheduled are placed below the horizontal axis and another unscheduled job is placed left of the vertical axis. This process is repeated until all jobs are scheduled.

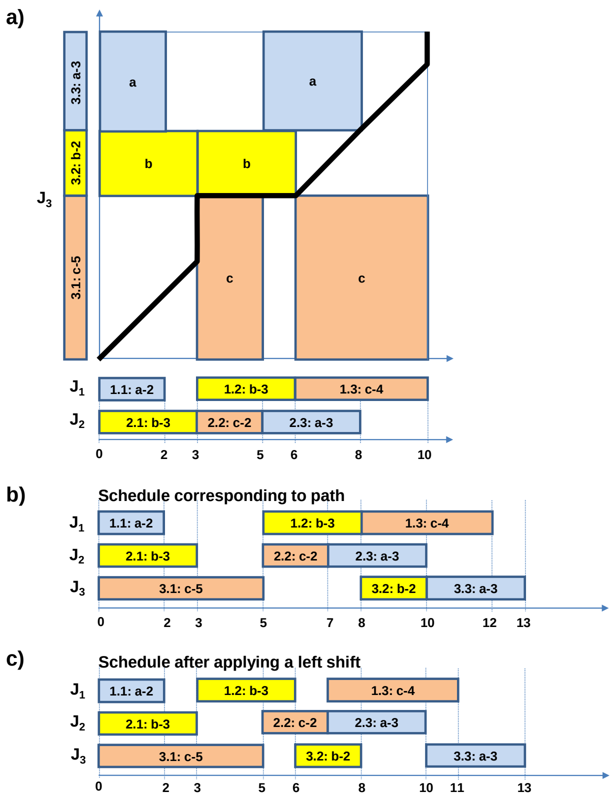

FIGURE 2.3. Example of the extension of the Akers graphical method for $n > 2$.

To decode the shortest path into the corresponding schedules of each job we follow the same rules used in the case $n = 2$. A horizontal segment implies that only the jobs in the horizontal axis are being processed, a vertical segment implied that only the job in the vertical axis is being processed, and a diagonal segment implies that all the jobs are being processed simultaneously. However, when $n > 2$ the following two problems may arise when applying the exact two-job graphical method:

(1) The shortest path obtained does not always correspond to a shortest path. This is so because when there is a vertical segment all the schedules of the jobs in the horizontal axis are delayed, which is not always necessary. To overcome this problem, we apply a left shift to all operations in the schedule (in a left shift, we move all operations in the schedule as far left as possible). Figure 2.3c illustrates the result of the application of a left shift to the schedule in Figure 2.3b.

(2) It may happen that adding the link (v, D_{NW}) when moving diagonally from a vertex v until an obstacle D is hit may lead to an invalid path segment going to the left (see Figure 2.4). To overcome this, we simply do not add to G links that correspond to path segments in the left direction.

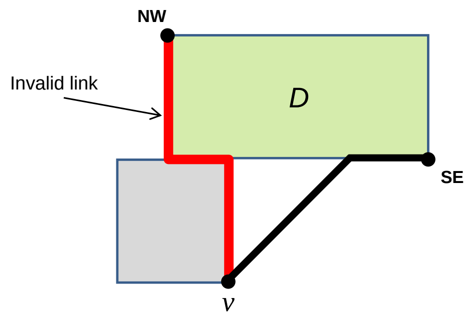

FIGURE 2.4. Invalid link.

Figure 2.5 presents pseudo code for the scheduling procedure AKERS_EXT which extends the graphical approach to the case $n > 2$. The procedure receives as input the set *SchedJobs* of jobs already scheduled, the current schedule *CurSch* of all jobs $j \in SchedJobs$, and the sequence $AddSeq = \{J_1, J_2, \ldots, J_n\}$ in which the jobs $j \notin SchedJobs$ will be added to schedule *CurSch*.

2.3. **New Local Search.** We next present a set of new local search algorithms for the JSP. Given a current schedule, we generate new schedules by removing nr jobs, apply a left-shift operator to all remaining operations, and add back the nr previously removed jobs using procedure AKERS_EXT, whose pseudo code is shown in Figure 2.5.

To illustrate how the new schedules are generated, we consider again the 4-job example given in Table 1. We use as the current schedule the initial schedule given in Figure 2.6a. The first step consists in removing from the schedule a number of jobs. We will remove jobs J_1 and J_4. We then apply a left shift to the resulting schedule and end up with the schedule shown in Figure 2.6b.

Next, the local search adds the removed jobs in the order given by *AddSeq* which we assume in this example to be $AddSeq = \{J_1, J_4\}$. To obtain the new solution all that is required is to run procedure AKERS_EXT with *CurSch* equal to the schedule given in Figure 2.6b, $AddSeq = \{J_1, J_4\}$, and $SchedJobs = \{J_2, J_3\}$. Figures 2.7 and 2.8 depict the Akers graph, the shortest path, and the corresponding schedules for jobs J_1, J_2, J_3 and J_1, J_2, J_3, J_4 after adding back job J_1 and after adding back jobs J_1 and J_4, respectively. Note that the new final schedule not only is different from the initial schedule but also has a smaller makespan.

Several variants of this local search algorithm can be produced by changing the number of jobs to be removed. As before, let *CurSch* denote the current schedule associated with the set of jobs J and let nr be the number of jobs to be removed. The corresponding flowchart of this new variant of the local search is shown in Figure 2.3.

```
procedure AKERS_EXT (CurSch, SchedJobs, AddSeq )
    if SchedJobs = {∅} then SchedJobs ← {AddSeq(1)}
    for s = 2 to n do
        Set J_add ← AddSeq(s)
        Construct an Akers graph where job J_add is placed in the
        vertical axis and all the jobs j ∈ SchedJobs are placed
        below the horizontal axis according to schedule CurSch.

        Find the shortest path in the Akers graph and assign
        the schedule of each job j ∈ SchedJobs ∪ {J_add}
        to schedule NewSch.

        Apply the left shift operator to schedule NewSch.

        // Update sets
        SchedJobs ← SchedJobs ∪ {J_add}
        CurSch ← NewSch
    end for
    return CurSch;
end AKERS_EXT;
```

FIGURE 2.5. Pseudo-code for the AKERS_EXT schedule construction procedure.

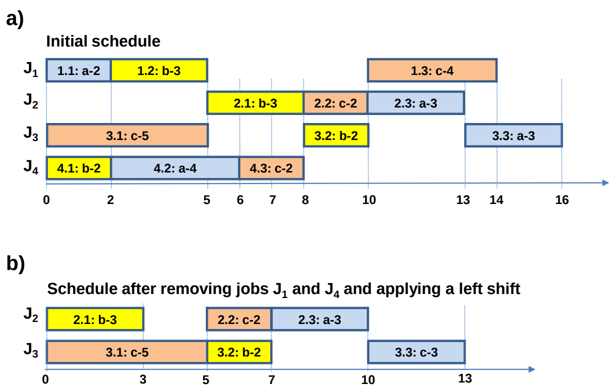

FIGURE 2.6. Removal of jobs J_1 and J_4 and left shifting the resulting scheduling.

Despite being very effective, the LS_AKERS_EXT local search procedure can have long running times when $nr \geq 2$. To overcome this problem, we propose a new variant of LS_AKERS_EXT where $nr \leq 2$. When $nr = 2$, each job $j \in J$ is combined with only $nRand$ jobs, chosen at random from the set $J \setminus \{j\}$. We call this new variant LS1+_AKERS_EXT and its corresponding pseudo-code is shown in Figure 2.10. Note that the LS1+_AKERS_EXT local search guarantees that every job $j \in J$ is removed from the schedule and is added back. Also, note that when $nRand = 0$ we obtain the LS_AKERS_EXT local search for the case where $nr = 1$. Likewise, when $nRand = n - 1$ we obtain the LS_AKERS_EXT local search for the case where $nr = 2$.

The heuristic AKERS_EXT runs $2 \times n \times nr$ times in the LS1+_AKERS_EXT local search and the complexity of AKERS_EXT is $O(nr \times n \times m \times log(n \times m))$. Therefore, the complexity of LS1+_AKERS_EXT is $O(n^2 \times m \times log(n \times m))$. Since $m = O(n)$, this complexity reduces to $O(n^3 \times log(n))$.

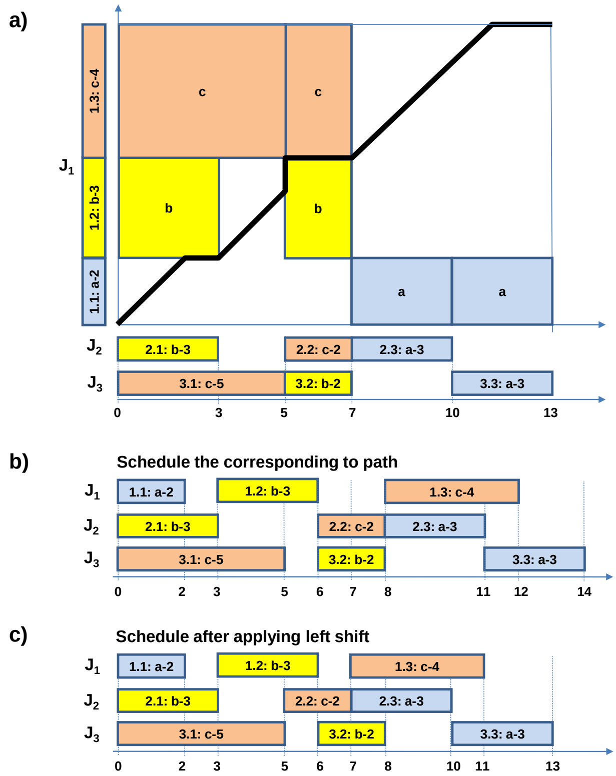

FIGURE 2.7. Schedule after adding back job J_1.

3. THE NEW HEURISTIC

The new heuristic proposed in this paper is a biased random-key genetic algorithm (BRKGA). In this section, we first briefly review the BRKGA framework. Then, we describe the encoding/decoding of the chromosome with a schedule generation scheme and an improvement procedure. We finally describe a chromosome adjustment procedure.

3.1. **Biased random-key genetic algorithm.** Genetic algorithms with random keys, or *random-key genetic algorithms* (RKGA), for solving optimization problems whose solutions can be represented as permutation vectors were introduced in Bean (1994). In a RKGA, chromosomes are represented as vectors of randomly generated real numbers in the interval $[0, 1]$. A deterministic algorithm, called a *decoder*, takes as input a solution vector and associates with it a solution of the combinatorial optimization problem for which an objective value or fitness can be computed.

A RKGA evolves a *population*, or set, of random-key vectors over a number of iterations, or *generations*. The initial population is made up of p vectors, each with $o = n \times m$ random keys. Each component of the solution vector, or random key, is generated independently at random in the real interval $[0, 1]$. After the fitness of each individual is computed by the decoder in generation k, the population is partitioned into two groups of individuals: a small group of p_e *elite* individuals, i.e. those with the best fitness values, and the remaining set of $p - p_e$ *non-elite* individuals. To evolve the population, a new generation of individuals must be produced. All elite individual of the population of generation g are copied without modification to the population of generation $g + 1$. RKGAs implement mutation by introducing *mutants* into the population. A mutant is simply a vector of random keys generated in the same way that an element of the initial population is generated. At each generation, a small number p_m of mutants is introduced into the population. With $p_e + p_m$ individuals accounted for in population $g + 1$, $p - p_e - p_m$ additional individuals need to be generated to complete the p individuals that make up population $g + 1$. This is done by producing $p - p_e - p_m$ offspring solutions through the process of *mating* or *crossover*.

A *biased random-key genetic algorithm*, or BRKGA (Gonçalves and Resende, 2011), differs from a RKGA in the way parents are selected for mating. While in the RKGA of Bean (1994) both

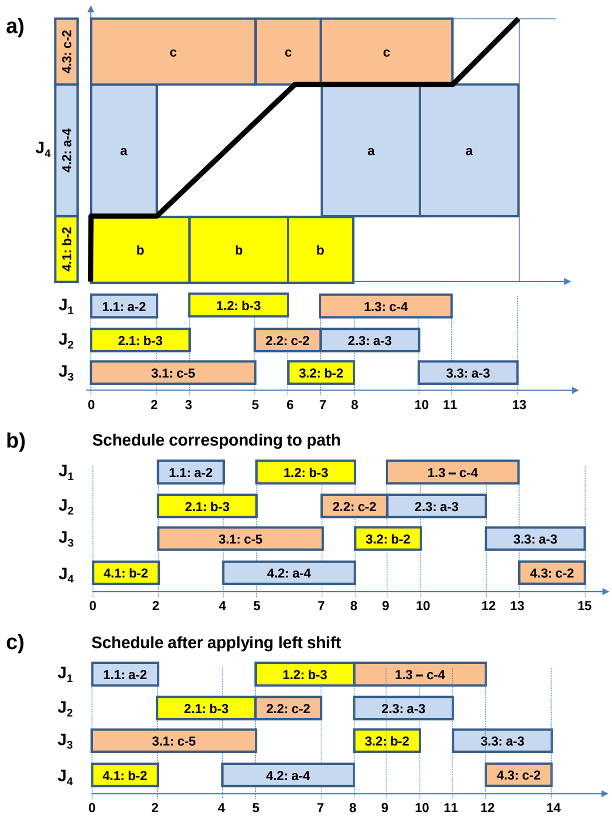

FIGURE 2.8. Schedule after adding back jobs J_1 and J_4.

parents are selected at random from the entire current population, in a BRKGA each element is generated combining a parent selected at random from the elite partition in the current population and one from the rest of the population. Repetition in the selection of a mate is allowed and therefore an individual can produce more than one offspring in the same generation. As in RKGAs, *parameterized uniform crossover* (DeJong and Spears, 1991) is used to implement mating in BRKGAs. Let ρ_e be the probability that an offspring inherits the vector component of its elite parent. Recall that o denotes the number of components in the solution vector of an individual. For $i = 1, \ldots, o$, the i-th component $c(i)$ of the offspring vector c takes on the value of the i-th component $e(i)$ of the elite parent e with probability ρ_e and the value of the i-th component $\bar{e}(i)$ of the non-elite parent $\bar{e}$ with probability $1 - \rho_e$.

When the next population is complete, i.e. when it has p individuals, fitness values are computed for all of the newly created random-key vectors and the population is partitioned into elite and non-elite individuals to start a new generation.

A BRKGA searches the solution space of the combinatorial optimization problem indirectly by searching the continuous o-dimensional hypercube, using the decoder to map solutions in the hypercube to solutions in the solution space of the combinatorial optimization problem where the fitness is evaluated.

To specify a biased random-key genetic algorithm, we simply need to specify how solutions are encoded and decoded. We specify our algorithm in the next section by first showing how schedules are encoded and then how decoding is done.

We have been building powerful heuristics based on the biased random-key genetic algorithm framework for over ten years (Gonçalves and Resende, 2011). We have observed that this framework allows the control and coordination of one or more heuristics enabling us to find solutions of much better quality than those found by the heuristics alone. The BRKGA works as a kind of long-term memory mechanism that learns how to best control the heuristic as the generations proceed. For example, in a set covering problem (Resende et al., 2012), the BRKGA controls a greedy algorithm by "learning" which sets are in a partial cover and only uses the greedy algorithm,

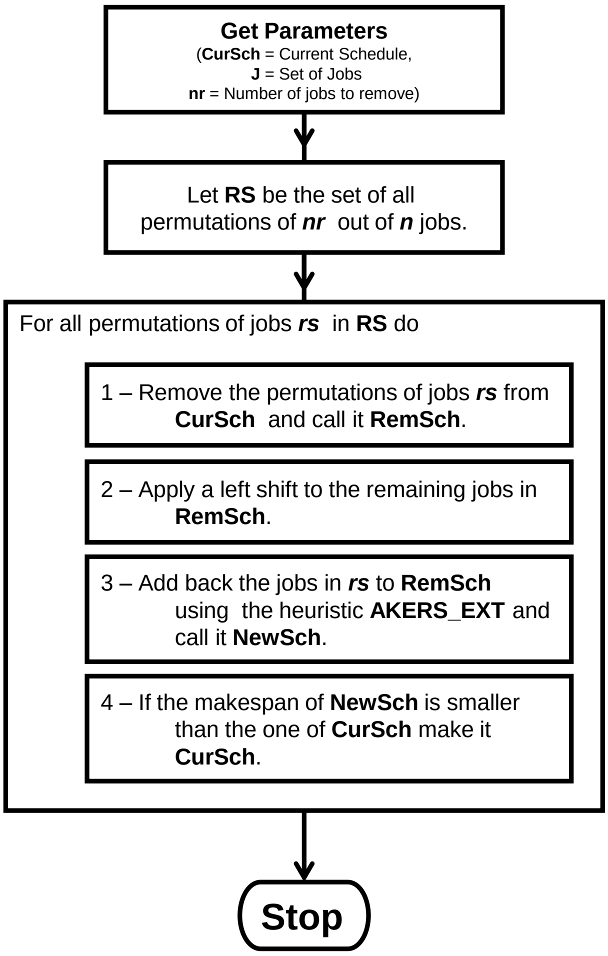

FIGURE 2.9. Flowchart for the LS_AKERS_EXT local search procedure.

starting from the "learned" partial cover, to complete the cover. In a 2D orthogonal packing problem (Gonçalves and Resende, 2011) where a number of small rectangles are packed in a large rectangle with the objective of maximizing the value of the packed rectangles, the BRKGA controls two simple heuristics (bottom-left and left-bottom) by "learning" the sequence the small rectangles are packed and which simple heuristic is used to pack each small rectangle. In the case of the job-shop scheduling problem, we expected that the BRKGA would learn a good order of the operations (and subsequent schedule) which could be improved by the local search heuristics employed here. As we will see in the remainder of this paper, the BRKGA does indeed achieve this goal.

3.2. **Solution Encoding.** We now describe the chromosome representation, i.e., how solutions to the problem are represented. The direct mapping of schedules as chromosomes is too complicated to represent and manipulate. In particular, it is difficult to develop corresponding crossover and mutation operations. As is always the case with BRKGAs, solutions (in this case schedules) are represented indirectly by parameters that are later used by a decoder to extract a solution. In this BRKGA, a schedule is represented by the following chromosome structure:

$$chromosome = (\underbrace{gene_1, ..., gene_{n_1}}_{n_1}, \underbrace{gene_{n_1+1}, ..., gene_{n_1+n_2}}_{n_2}, \; \cdots \;, \underbrace{gene_{o-n_n+1}, ..., gene_o}_{n_n})$$

where n_j represents the number of operations of job $j = 1, ..., n$. Each gene is a randomly generated real number in the interval $[0, 1]$. The value of each gene is used in the decoding procedure described in the next subsection.

```
procedure LS1+_AKERS_EXT (CurSch, J, nRand )
    for j = 1 to n do
        Choose randomly nRand jobs from the set J \ {j}
        and assign them to set RandJobs;

        if nRand > 0 then
            Let RS be the set of all ordered combinations of two
            jobs where one job is j and the other belongs
            to the set RandJobs;
        else
            Let RS be {j};
        end if

        for all rs ∈ RS do
            // Update set of scheduled jobs
            SchedJobs ← J \ rs;

            // Remove the jobs in rs and apply a left shift
            Let RemSch be the schedule of all the jobs
            j ∈ SchedJobs obtained after removing from schedule
            CurSch all the jobs j ∈ rs and applying a left shift;

            // Add back the jobs in the order given by rs
            NewSch ← AKERS_EXT(RemSch, SchedJobs, rs);

            // If makespan is reduced update current schedule
            if makespan(NewSch) < makespan(CurSch) then
                CurSch ← NewSch;
            end if
        end for
    end for

    return CurSch;
end LS1+_AKERS_EXT;
```

FIGURE 2.10. Pseudo-code for the `LS1+_AKERS_EXT` local search procedure.

3.3. **Decoding a random-key vector into a job-shop schedule.** The decoding process of a chromosome into a schedule consists of three steps: initial schedule generation; local search with tabu search; and chromosome adjustment. We next describe each of these components. Figure 3.1 illustrates the sequence of steps applied to each chromosome in the decoding process.

3.3.1. *Initial schedule generation.* An initial schedule is decoded from a chromosome with the following two steps:

(1) Translate the chromosome into a list of ordered operations;
(2) Generate the schedule with a one-pass heuristic based on the list obtained in (1).

To translate the chromosome, we use an operation-based representation where a schedule is represented by an unpartitioned permutation with n_j repetitions of each job j (Gen et al., 1994, Bierwirth, 1995, Cheng et al., 1996, Shi et al., 1996). Because of the precedence constraints, each repeating gene does not indicate a concrete operation of a job but refers to a unique operation which is context-dependent. To illustrate the translation process we will use the example in Table 1. The process starts by filling an unordered vector of jobs with the number of each job repeated n_j times (see Figure 3.2a). Next, the vector is ordered according to the values of the corresponding genes in the chromosome (see Figure 3.2b). Finally, the list of ordered operations is obtained by replacing, from left to right, each k^{th} job number occurrence in the ordered vector of jobs by the k^{th} operation in the technological sequence of the job (see Figure 3.2c).

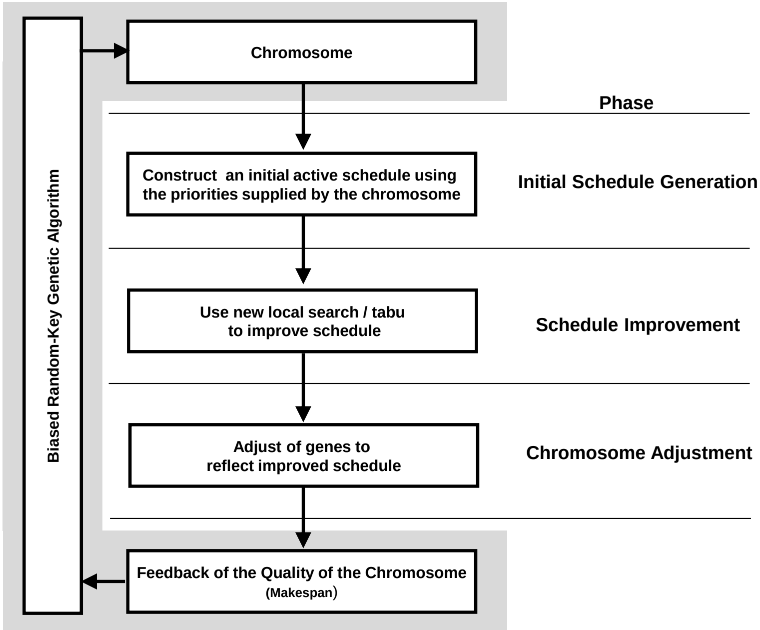

FIGURE 3.1. Sequence of steps applied to each chromosome in the decoding process.

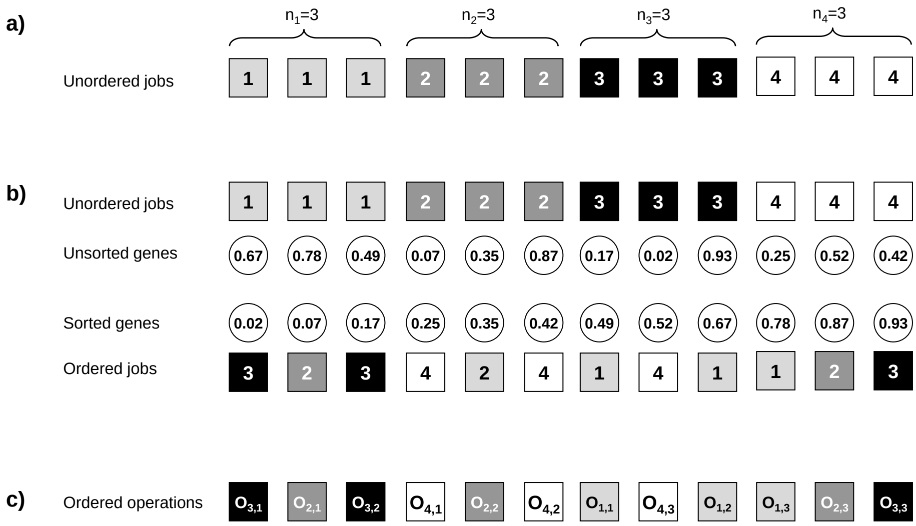

FIGURE 3.2. Translating a chromosome into a list of ordered operations.

Once a list of ordered operations is obtained, a schedule is constructed by initially scheduling the first operation in the list, then the second operation, and so on. Each operation is assigned to the earliest feasible starting time in the machine it requires. The process is repeated until all operations are scheduled (see Figure 3.3 for the final schedule corresponding to the ordered operation list in Figure 3.1c). Note that the schedules generated by this process are guaranteed to be active schedules. An *active schedule* is one where no activity can be started earlier without changing the start times of any other activity and still maintain feasibility (Schrage, 1970).

3.3.2. *Local search with tabu search.* After an initial schedule using the random keys provided by the BRKGA is obtained and the decoding procedure described in the previous section is carried out, we proceed by trying to improve the schedule with a new hybrid local search that we developed. This new local search, denoted by NEW_LS, combines the LS1+_AKERS_EXT local search (introduced in Section 2.3) with a tabu search procedure that uses the neighborhood structure

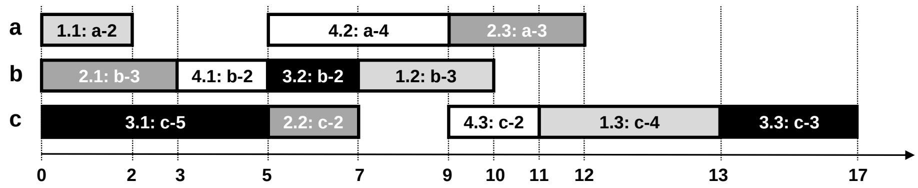

FIGURE 3.3. Initial active schedule obtained from the list of ordered operations.

proposed by Nowicki and Smutnicki (1996) and will be denoted as TS_NS. The neighborhood of Nowicki and Smutnicki randomly selects a critical path in the current schedule and identifies all of its critical blocks (sequences of contiguous operations on the same machine). Then, it considers for exchange only the first two and the last two operations in every block (the first two and last two operations in the critical path are excluded). To select a move, we must first evaluate the makespan of every move in the neighborhood. Since the exact evaluation of a move is time consuming, we use the fast approximate method of Taillard (1994) in place of the exact evaluation. The move with the smallest approximate makespan is selected and applied. We then compute the exact makespan. To do that, we use the topological order and the efficient updating procedures for heads and tails of Nowicki and Smutnicki (2005).

The TS_NS tabu search is embedded into the LS1+_AKERS_EXT local search between lines 9 and 10 of its pseudo-code.

The tabu list, TL, consists of $maxT$ operation pairs that have been exchanged in the last $maxT$ moves of the tabu search. If the move corresponding to the exchange of the operations in pair $\{o_u, o_v\}$ has been performed then its inverse pair $\{o_v, o_u\}$ replaces the oldest move in TL (or is added to the end of the list TL if it is not full). This process prevents the exchange of the same operations for the next $maxT$ moves.

The pseudo-code for the `TS_NS` hybrid local search procedure is depicted in Figure 4.1.

3.3.3. *Chromosome Adjustment.* Solutions produced by the hybrid local search procedure `NEW_LS` usually disagree with the genes initially supplied to the decoder in the vector of random keys. Changes in the order of the operations made by the local search phase of the decoder need to be taken into account in the chromosome. The heuristic adjusts the chromosome to reflect these changes. To make the chromosome supplied by the GA agree with the solution produced by local search, the heuristic adjusts the order of the genes according to the starting times of the operations. This chromosome adjustment not only improves the quality of the solutions but also reduces the number of generations needed to obtain the best values.

3.4. **Fitness measure.** A natural fitness function (measure of quality) for this type of problem is C_{max}. However, since different schedules can have the same makespan, this measure does not differentiate well the potential for improvement of schedules having identical makespans. To better differentiate the potential for improvement we used a measure called *modified makespan* that is detailed in Mendes et al. (2009) and Gonçalves et al. (2010). The modified makespan combines the makespan of the schedule with a measure of the potential for improvement of the schedule which has values in the interval $]0, 1[$. The rationale for this new measure is that if we have two schedules with the same makespan value, then the one with a smaller number of activities ending close to the makespan will have more potential for improvement.

4. Experimental results

We now report results obtained on a set of experiments conducted to evaluate the performance of *BRKGA-JSP,* the algorithm proposed in this paper. *BRKGA-JSP* was implemented in C++ and all the computational experiments were carried out on a computer with an AMD 2.2 GHz Opteron (2427) CPU running the Linux (Fedora release 12) operating system. We list the benchmark instances and algorithms used in the experiments, specify the parameter configuration used in the experiments, and describe the results.

```
procedure TS_NS (CurSch)
    Let BestSch be the best schedule found in the procedure;
    Let nNIM be the number of non improving moves;
    Let TL be a tabu list with length maxT;
    Let maxNIM be the maximum number allowed of consecutive
        non improving moves;
    nNIM ← 0;
    BestSch ← CurSch;
    Continue ← TRUE; // used to stop while loop;
    while (nNIM ≤ maxNIM and Continue) do

        Let CP be a critical path in the current schedule CurSch;
        Let NS be the set operations pairs generated by the neighborhood of
            Nowicki and Smutnicki when applied to critical path CP.
        Evaluate the makespan corresponding to each move in NS using
            the approximate method of Taillard (1994);

        Let DS be the set operations pairs in NS which correspond to
            moves that decrease the makespan
        Let IS be the set operations pairs in NS \ TL which correspond to
            moves that increase the makespan;

        if (DS = {∅} and IS = {∅}) then
            Continue ← FALSE; // stop search;
        else
            if DS ≠ {∅} then;
                Let {o_u, o_v} be the operation pair in DS that decreases the
                    most the makespan;
            else
                Let {o_u, o_v} be the operation pair in IS that increases the
                    least the makespan;
            end if
            Exchange operations in {o_u, o_v} and update the schedule
                and its makespan using the exact procedures from
                Nowicki and Smutnicki (2005). Denote the resulting
                schedule as NewSch;

            Update TL with the operation pair {o_v, o_u};

            if makespan(NewSch) < makespan(BestSch) then
                BestSch ← NewSch;
                nNIM ← 0;
            else
                nNIM ← nNIM + 1;
            end if

            CurSch ← NewSch;
        end if

    end while

    return BestSch;
end TS_NS;
```

FIGURE 4.1. Pseudo-code for the TS_NS tabu search procedure.

4.1. **Benchmark instances and algorithms.** To illustrate the effectiveness of *BRKGA-JSP*, we consider the following well-known problem classes from the literature :

- FT – 3 problems denoted as (FT06, FT10 and FT20) due to Fisher and Thompson (1963);
- LA – 40 problems denoted as (LA01 – LA40) due to Lawrence (1984);
- ABZ – 3 problems denoted as (ABZ07 – ABZ09) due to Adams et al. (1988)
- ORB – 10 problems denoted as (ORB01 – ORB10) due to Applegate and Cook (1991);
- YN – 4 problems denoted as (YN01 – YN04) due to Yamada and Nakano (1992)
- SWV – 15 problems denoted as (SWV01 – SWV15) due to Storer et al. (1992)
- TA – 50 problems denoted as (TA01 – TA50) due to Taillard (1994). Instances TA51–80 are commonly considered easy and the corresponding results are not usually reported. Since *BRKGA-JSP* obtained the optimal solutions to all these instances, we will focus our attention only on the instances TA01–50 which are more difficult.
- DMU – 80 problems denoted as (DMU01 – DMU80) due to Demirkol et al. (1997).

We compare our results with those obtained by the currently best performing approaches found in the literature, namely:

- i-TSAB (Nowicki and Smutnicki, 2005);
- GES (Pardalos and Shylo, 2006);
- TS (Zhang et al., 2007);
- TS/SA (Zhang et al., 2008);
- AlgFix (Pardalos et al., 2010).

4.2. **Configuration.** All the computational experiments were conducted using the same configuration parameters shown in Table 2.

TABLE 2. Configuration parameters.

Parameter	Value
BRKGA	
p	$max\,(\,150,\ \lceil 0.5 \times o \rceil\,)$
p_e	10
p_m	10
ρ_e	0.85
Fitness	Modified makespan (to minimize)
Stopping Criterion	20 generations
LS1+_AKERS_EXT	
nRand	$max\,(\,4,\ min\,(\,\lceil 0.3 \times n \rceil\,,\,12)\,)$
TS_NS	
$maxNIM$	100
$maxT$	$max\,(\,4,\ \lceil 0.3 \times n \rceil\,)$
Number of runs	10

$\lceil x \rceil$ denotes the smallest integer greater than x

4.3. **Results.** To compare with other approaches we use the following measures:

$\%RE =$ the % relative error of a solution with makespan C_{max} with respect to the best-known upper bound (UB), i.e.,
$\%RE = 100\% \times (C_{max} - UB)\,/UB$.

$\%ARE =$ average $\%RE$ over all instances.

Because some of the literature describing other approaches with which we compare our heuristic do not report detailed results for each instance or report results relative to best known values that are not reported, we only compute %ARE for *BRKGA-JSP* using those instances reported in detail in the literature. The values for which there are no detailed information are left blank in our tables. For all instances we provide its lower bound (LB) and and best known value (UB) (when $LB = UB$ the best known value is optimal). The updated values of LB and UB were obtained from the following papers: Taillard (1994), Balas and Vazacopoulos (1998), Wennink (1995), Nowicki and Smutnicki (1996), Vaessens et al. (1996), Demirkol et al. (1997), Jain (1998), Brinkkötter and Brucker (2001), Schilham. (2001), Henning (2002), Nowicki and Smutnicki (2002), Pardalos and Shylo (2006), Zhang et al. (2008), Pardalos et al. (2010) and the URLs: `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/jobshop.dir/best_lb_up.txt`,

`http://plaza.ufl.edu/shylo/TA.html`, and `http://plaza.ufl.edu/shylo/DMU.html`.

The detailed experimental results obtained for the problem classes FT, ORB, LA, ABZ, YN, TA, and DMU are presented in Tables 7 to 15 in the appendix. Note that since not all the other approaches report results for the same set of instances, we have to use two rows with labels %ARE and BRKGA-JSP %ARE at the bottom of some tables to aggregate the average $\%RE$ over all instances being compared.

Optimal solutions for all the instances in problem classes FT, ORB, and LA are known. For problem classes FT and ORB, the approaches BRKGA-JSP, GES, TS, and TS/SA obtained optimal solutions on all instances. For problem class LA, the approach GES obtained the optimal solutions on all instances, while BRKGA-JSP failed to do so on instance LA29 where it obtained a value of 1153 instead of 1152. Approaches TS and TS/SA failed to find optimal values for instances LA29 and LA40. Problem classes ABZ, YN, TA, and DMU include some hard instances for which no optimal solution is known. BRKGA-JSP obtained the best %ARE results for these classes with the exception of problem class SWV where the TS approach, which only presents values for instances SWV11-15, obtained a single better result (for instance SWV15). BRKGA-JSP improved the best known values (UB) for 57 instances (42 on the DMU class, nine on the TA class, one on the YN class, and five on the SWV class). Table 3 presents a summary of the %ARE obtained by each approach for each problem class (note that since not all the other approaches use the same set of instances we have to use two rows for each class of problems - the row that starts with “other” presents the results obtained by the other approaches and the row starting with BRKGA-JSP present the results for the corresponding instance obtained by our algorithm).

TABLE 3. Summary of %ARE obtained by each approach for each instance class.

Class	Approach	GES	TS	TS/SA	AlgFix	*i*-TSAB
FT	Other		**0**	**0**		
	BRKGA-JSP		**0**	**0**		
ORB	Other	**0**		**0**		
	BRKGA-JSP	**0**		**0**		
LA	Other	**0.000**	0.046	0.023		
	BRKGA-JSP	0.002	**0.008**	**0.008**		
ABZ	Other		0.350	0.202		
	BRKGA-JSP		**0.100**	**0.100**		
YN	Other			0.026		
	BRKGA-JSP			**-0.083**		
SWV01-10	Other			0.007		
	BRKGA-JSP			**-0.015**		
SWV11-15	Other		**0.000**			
	BRKGA-JSP		0.010			
TA	Other	0.194		0.119	0.518	0.194
	BRKGA-JSP	**-0.023**		**-0.023**	**-0.023**	**-0.043**
DMU	Other	0.629	0.162		0.424	1.150
	BRKGA-JSP	**-0.104**	**-0.155**		**-0.104**	**-0.138**

Best values of %ARE are in **bold**.

To investigate the contribution of each of the components included in BRKGA-JSP (Genetic Algorithm, Tabu Search, `LS1+_AKERS_EXT`, and Chromosome Adjustment) we conducted the additional experiments using the components described in Table 4.

TABLE 4. Description of additional experiments.

Experiment	Description
GA	Run BRKGA alone, using chromosome adjustment.
GA-TS	Run BRKGA with Tabu Search and chromosome adjustment.
GA-AK	Run BRKGA with the `LS1+_AKERS_EXT` search and chromosome adjustment.
GA-AKTS	Run BRKGA with both `LS1+_AKERS_EXT` search and Tabu Search, but without chromosome adjustment.

Table 5 lists, for each problem class, %GA, %GA-TS, % GA-AK, and %GA-AKTS, the average % increase in makespan for GA, GA-TS, GA-AK, and GA-AKTS, respectively, with respect to the average makespans of the solutions obtained by BRKGA-JSP.

TABLE 5. % increase in makespan with respect to full algorithm for each experiment on all instance classes.

Class	$n \times m$	% GA	% GA-TS	% GA-AK	% GA-AKTS
FT06	6×6	0.0%	0.0%	0.0%	0.0%
FT10	10×10	5.9%	0.0%	0.9%	0.0%
FT20	20×5	6.8%	0.7%	0.0%	0.0%
ORB01-10	10×10	7.0%	0.6%	0.3%	0.0%
LA01-05	10×5	0.9%	0.0%	0.0%	0.0%
LA06-10	15×5	0.0%	0.0%	0.0%	0.0%
LA11-15	20×5	0.0%	0.0%	0.0%	0.0%
LA16-20	10×10	2.2%	0.1%	0.1%	0.0%
LA21-25	15×10	7.0%	0.3%	1.1%	0.0%
LA26-30	20×10	7.6%	0.9%	1.4%	0.2%
LA31-35	30×10	0.2%	0.0%	0.0%	0.0%
LA36-40	15×15	11.2%	1.3%	1.6%	0.0%
ABZ07-09	20×15	14.7%	2.3%	3.1%	0.3%
YN01-04	20×20	13.6%	1.8%	3.6%	0.5%
SWV01-05	20×10	19.9%	6.3%	3.8%	0.6%
SWV06-10	20×15	22.9%	6.9%	6.6%	1.5%
SWV11-15	50×10	28.8%	10.9%	7.2%	0.6%
TA01-10	15×15	10.3%	0.8%	1.3%	0.0%
TA11-20	20×15	14.6%	2.7%	3.8%	0.5%
TA21-30	20×20	14.9%	1.9%	4.4%	0.5%
TA31-40	30×15	15.0%	2.4%	6.6%	0.6%
TA41-50	30×20	20.7%	4.0%	9.3%	1.4%
DMU01-05	20×15	17.6%	1.8%	3.6%	0.4%
DMU06-10	20×20	17.3%	1.9%	3.7%	0.3%
DMU11-15	30×15	16.4%	2.7%	6.9%	0.5%
DMU16-20	30×20	18.6%	3.0%	8.7%	0.7%
DMU21-25	40×15	7.8%	0.0%	2.3%	0.0%
DMU26-30	40×20	16.8%	2.2%	8.0%	0.3%
DMU31-35	50×15	3.7%	0.0%	1.2%	0.0%
DMU36-40	50×20	14.3%	1.0%	6.1%	0.0%
DMU41-45	20×15	22.8%	7.2%	6.4%	1.5%
DMU46-50	20×20	22.3%	5.9%	7.9%	1.4%
DMU51-55	30×15	28.8%	10.1%	9.5%	2.1%
DMU56-60	30×20	28.7%	11.4%	11.7%	3.0%
DMU61-65	40×15	31.5%	13.8%	11.1%	0.2%
DMU66-70	40×20	32.1%	13.3%	13.5%	0.1%
DMU71-75	50×15	33.1%	15.3%	12.5%	0.1%
DMU76-80	50×20	35.0%	17.7%	14.2%	0.1%
	Overall average =	15.0%	4.0%	4.8%	0.5%

From Table 5 it is clear that the GA alone does not perform well since it produces an overall average increase of 15% in the makespans. The combinations of the GA with the tabu search (GA-TS) and with the `LS1+_AKERS_EXT` (GA-AK) produce better results that are only 4% to 5% above the ones produced by BRKGA-JSP. The combination of the GA with both the `LS1+_AKERS_EXT` search and the tabu search (GA-AKTS) results in the best makespans of the four, and is only 0.5% above the average makespan of the solutions found by BRKGA-JSP. This shows that the use of chromosome adjustment is important since it produces an additional 0.5% in makespan reduction.

In terms of computational times, we cannot make any fair and meaningful comment since all the other approaches were implemented with different programming languages and tested on computers with different computing power. Hence, to avoid discussion about the different computers speed used in the tests, we limit ourselves to reporting in Table 6 the average running times per run for BRKGA-JSP, while for each of the other algorithms we only report, when available, the CPU used and the reported running times. We profiled our runs and also include the percentage of the total time that was spent on each of the algorithm components of BRKGA-JSP (%GA – genetic algorithm, %TS – tabu search, and %AK – `LS1+_AKERS_EXT` search). It is clear from Table 6 that BRKGA-JSP spends most of its time in the `LS1+_AKERS_EXT` search.

TABLE 6. Average running times for BRKGA-JSP.

		BRKGA-JSP				i-TSAB	TS	TS/SA	AlgFix	GES
Class	$n \times m$	% GA	% TS	% AK	Time(s)	Time(s)	Time(s)	Time(s)	Time(s)	Time(s)
FT06	6 × 6	12.50%	25.00%	62.50%	1.0					
FT10	10 × 10	4.44%	6.67%	88.89%	10.1		41.1	3.8		
FT20	20 × 5	1.75%	1.75%	96.49%	13.4					
ORB01-10	10 × 10	1.73%	2.80%	95.47%	5.8			6.2		
LA01-05	10 × 5	7.04%	10.80%	82.15%	1.4			0.0		
LA06-10	15 × 5	3.34%	3.92%	92.74%	2.9					
LA11-15	20 × 5	1.86%	1.93%	96.21%	5.3					
LA16-20	10 × 10	5.92%	7.92%	86.16%	4.6			0.2		
LA21-25	15 × 10	1.93%	3.17%	94.90%	15.3			13.6		
LA26-30	20 × 10	0.95%	1.51%	97.54%	21.8			15.2		
LA31-35	30 × 10	0.31%	0.49%	99.21%	38.7					
LA36-40	15 × 15	1.73%	2.81%	95.46%	21.4			36.1		
ABZ07-09	20 × 15	1.17%	1.76%	97.07%	54.6			88.9		
YN01-04	20 × 20	0.90%	2.01%	97.10%	105.2			109.1		
SWV01-05	20 × 10	0.71%	1.52%	97.78%	42.5			138.3		
SWV06-10	20 × 15	0.76%	1.46%	97.79%	78.7			190.2		
SWV11-15	50 × 10	0.69%	1.87%	97.44%	2304.4		3118.2			
TA01-10	15 × 15	0.48%	1.17%	98.35%	30.4	79		65.3	10000	30000
TA11-20	20 × 15	0.18%	0.61%	99.21%	65.8	390		235	10000	30000
TA21-30	20 × 20	2.48%	3.93%	93.59%	143.2	1265		433	10000	30000
TA31-40	30 × 15	3.12%	3.92%	92.96%	487.6	1225		370.4	10000	30000
TA41-50	30 × 20	0.31%	0.69%	99.01%	1068.3	1670		845.8	10000	30000
DMU01-05	20 × 15	1.04%	1.51%	97.45%	68.9				10000	30000
DMU06-10	20 × 20	0.92%	1.78%	97.31%	145.4				10000	30000
DMU11-15	30 × 15	0.25%	0.51%	99.25%	427.3				10000	30000
DMU16-20	30 × 20	0.21%	0.72%	99.07%	1043.6				10000	30000
DMU21-25	40 × 15	0.09%	0.28%	99.64%	1150.6				10000	30000
DMU26-30	40 × 20	0.08%	0.37%	99.55%	3556.3				10000	30000
DMU31-35	50 × 15	0.08%	0.18%	99.74%	2086.7				10000	30000
DMU36-40	50 × 20	0.05%	0.24%	99.71%	9368.3				10000	30000
DMU41-45	20 × 15	0.56%	1.21%	98.23%	78.9				10000	30000
DMU46-50	20 × 20	0.52%	1.42%	98.06%	187.7				10000	30000
DMU51-55	30 × 15	0.16%	0.49%	99.35%	701.4				10000	30000
DMU56-60	30 × 20	0.14%	0.63%	99.23%	1545.8				10000	30000
DMU61-65	40 × 15	0.07%	0.28%	99.65%	2684.3				10000	30000
DMU66-70	40 × 20	0.07%	0.39%	99.54%	5394.2				10000	30000
DMU71-75	50 × 15	0.04%	0.21%	99.74%	8070.1				10000	30000
DMU76-80	50 × 20	0.04%	0.27%	99.69%	15923.4				10000	30000

Note. i-TSAB was run on a Pentium at 900 MHz, TS was run on a Pentium IV at 1.8 GHz, TS/SA was run on a Pentium IV at 3.0 GHz and AlgFix and GES were run on a Pentium at 2.8 GHz.

5. CONCLUDING REMARKS

This paper proposed a new local search for the job-shop scheduling problem and its application within a biased random-key genetic algorithm. Schedules are constructed using the chromosome supplied by the genetic algorithm and a procedure that generates active schedules. After a schedule is obtained, the new local search heuristic, based on an extension of the graphical approach of Akers (1956), is applied to improve the solution. The approach is tested on a set of 205 standard instances from the literature and compared with other approaches. The new heuristic improved the best known values for 57 instances. Overall, the experimental results validate the effectiveness of the proposed algorithm.

6. Appendix

Table 7. Makespan and average percent deviations from best upper bound for problem class FT.

			BRKGA-JSP			TS/SA	TS
Prob	$n \times m$	*Opt.*	max	avg	min	min	min
FT06	6×6	55	55	55	55		
FT10	10×10	930	930	930	930	930	930
FT20	20×5	1165	1165	1165	1165		
				%ARE =	0	0	0

Table 8. Makespan and average percent deviations from best upper bound for problem class LA.

			BRKGA-JSP			GES	TS/SA	TS
Prob	$n \times m$	*Opt.*	max	avg	min	min	min	min
LA01	10×5	666	666	666	666	666		
LA02	10×5	655	655	655	655	655		
LA03	10×5	597	597	597	597	597		
LA04	10×5	590	590	590	590	590		
LA05	10×5	593	593	593	593	593		
LA06	15×5	926	926	926	926	926		
LA07	15×5	890	890	890	890	890		
LA08	15×5	863	863	863	863	863		
LA09	15×5	951	951	951	951	951		
LA10	15×5	958	958	958	958	958		
LA11	20×5	1222	1222	1222	1222	1222		
LA12	20×5	1039	1039	1039	1039	1039		
LA13	20×5	1150	1150	1150	1150	1150		
LA14	20×5	1292	1292	1292	1292	1292		
LA15	20×5	1207	1207	1207	1207	1207		
LA16	10×10	945	945	945	945	945		
LA17	10×10	784	784	784	784	784		
LA18	10×10	848	848	848	848	848		
LA19	10×10	842	842	842	842	842	842	842
LA20	10×10	902	902	902	902	902		
LA21	15×10	1046	1046	1046	1046	1046	1046	1046
LA22	15×10	927	927	927	927	927		
LA23	15×10	1032	1032	1032	1032	1032		
LA24	15×10	935	935	935	935	935	935	935
LA25	15×10	977	977	977	977	977	977	977
LA26	20×10	1218	1218	1218	1218	1218		
LA27	20×10	1235	1235	1235	1235	1235	1235	1235
LA28	20×10	1216	1216	1216	1216	1216		
LA29	20×10	1152	1160	1154.7	1153	1152	1153	1156
LA30	20×10	1355	1355	1355	1355	1355		
LA31	30×10	1784	1784	1784	1784	1784		
LA32	30×10	1850	1850	1850	1850	1850		
LA33	30×10	1719	1719	1719	1719	1719		
LA34	30×10	1721	1721	1721	1721	1721		
LA35	30×10	1888	1888	1888	1888	1888		
LA36	15×15	1268	1268	1268	1268	1268	1268	1268
LA37	15×15	1397	1397	1397	1397	1397	1397	1397
LA38	15×15	1196	1196	1196	1196	1196	1196	1196
LA39	15×15	1233	1233	1233	1233	1233	1233	1233
LA40	15×15	1222	1226	1223.2	1222	1222	1224	1224
					%ARE =	0.000	0.023	0.046
			BRKGA-JSP		%ARE =	0.002	0.008	0.008

TABLE 9. Makespan and average percent deviations from best upper bound for problem class ORB.

				BRKGA-JSP			TS/SA	GES
Prob	n	m	*Opt.*	max	avg	min	min	min
ORB01	10	10	1059	1059	1059	1059	1059	1059
ORB02	10	10	888	888	888	888	888	888
ORB03	10	10	1005	1005	1005	1005	1005	1005
ORB04	10	10	1005	1011	1006.2	1005	1005	1005
ORB05	10	10	887	887	887	887	887	887
ORB06	10	10	1010	1010	1010	1010	1010	1010
ORB07	10	10	397	397	397	397	397	397
ORB08	10	10	899	899	899	899	899	899
ORB09	10	10	934	934	934	934	934	934
ORB10	10	10	944	944	944	944	944	944
					%ARE =	0	0	0

TABLE 10. Makespan and average percent deviations from best upper bound for problem class ABZ.

				BRKGA-JSP			TS/SA	TS
Prob	$n \times m$	*LB*	*UB*	max	avg	min	min	min
ABZ07	20 × 15	656	656	661	658	656	658	657
ABZ08	20 × 15	645	665	668	667.7	667	667	669
ABZ09	20 × 15	661	678	681	678.9	678	678	680
					%ARE =	0.100	0.202	0.350

TABLE 11. Makespan and average percent deviations from best upper bound for problem class YN.

				BRKGA-JSP			TS/SA
Prob	$n \times m$	*LB*	*UB*	max	avg	min	min
YN01	20 × 20	826	884	889	886	884	884
YN02	20 × 20	861	907	909	906.5	**904**	907
YN03	20 × 20	827	892	895	893.1	892	892
YN04	20 × 20	918	968	979	973	968	969
					%ARE =	-0.083	0.026

Newly found upper bounds by BRKGA-JSP are in **bold**.

TABLE 12. Makespan and average percent deviations from best upper bound for problem class SWV.

				BRKGA-JSP			TS/SA	TS
Prob	$n \times m$	*LB*	*UB*	max	avg	min	min	min
SWV01	20 × 10	1407	1407	1413	1408.9	1407	1412	
SWV02	20 × 10	1475	1475	1490	1478.2	1475	1475	
SWV03	20 × 10	1369	1398	1404	1400	1398	1398	
SWV04	20 × 10	1450	1470	1478	1472.8	1470	1470	
SWV05	20 × 10	1424	1424	1441	1431.4	1425	1425	
SWV06	20 × 15	1591	1678	1694	1682.1	**1675**	1679	
SWV07	20 × 15	1446	1600	1609	1601.2	**1594**	1603	
SWV08	20 × 15	1640	1756	1770	1764.3	**1755**	1756	
SWV09	20 × 15	1604	1661	1675	1667.9	**1656**	1661	
SWV10	20 × 15	1631	1754	1772	1754.6	**1743**	1754	
SWV11	50 × 10	2983	2983	2989	2985.9	2983		2983
SWV12	50 × 10	2972	2979	2994	2989.7	2979		2979
SWV13	50 × 10	3104	3104	3140	3111.6	3104		3104
SWV14	50 × 10	2968	2968	2968	2968	2968		2968
SWV15	50 × 10	2885	2886	2904	2902.9	2901		2886
						%ARE =	0.007	0.000
					BRKGA-JSP	%ARE =	-0.015	0.010

Newly found upper bounds by BRKGA-JSP are in **bold**.

TABLE 13. Makespan and average percent deviations from best upper bound for problem class TA.

				BRKGA-JSP			GES	AlgFix	i-TSAB	TS/SA
Prob	$n \times m$	LB	UB	max	avg	min	min	min	min	min
TA01	15 × 15	1231	1231	1231	1231	1231	1231	1231		1231
TA02	15 × 15	1244	1244	1244	1244	1244	1244	1244		1244
TA03	15 × 15	1218	1218	1218	1218	1218	1218	1218		1218
TA04	15 × 15	1175	1175	1175	1175	1175	1175	1175		1175
TA05	15 × 15	1224	1224	1227	1224.9	1224	1224	1224		1224
TA06	15 × 15	1238	1238	1240	1238.9	1238	1238	1238		1238
TA07	15 × 15	1227	1227	1228	1228	1228	1228	1228		1228
TA08	15 × 15	1217	1217	1217	1217	1217	1217	1217		1217
TA09	15 × 15	1274	1274	1280	1277	1274	1274	1274		1274
TA10	15 × 15	1241	1241	1241	1241	1241	1241	1241		1241
TA11	20 × 15	1323	1357	1365	1360	1357	1357	1358	1361	1359
TA12	20 × 15	1351	1367	1376	1372.6	1367	1367	1367		1371
TA13	20 × 15	1282	1342	1351	1347.3	1344	1344	1342		1342
TA14	20 × 15	1345	1345	1345	1345	1345	1345	1345		1345
TA15	20 × 15	1304	1339	1360	1348.9	1339	1339	1339		1339
TA16	20 × 15	1302	1360	1371	1362.1	1360	1360	1360		1360
TA17	20 × 15	1462	1462	1478	1470.5	1462	1469	1473	1462	1464
TA18	20 × 15	1369	1396	1407	1400.9	1396	1401	1396		1399
TA19	20 × 15	1297	1332	1338	1333.2	1332	1332	1332	1335	1335
TA20	20 × 15	1318	1348	1357	1350.4	1348	1348	1348	1351	1350
TA21	20 × 20	1539	1643	1650	1647	**1642**	1647	1643	1644	1644
TA22	20 × 20	1511	1600	1600	1600	1600	1602	1600	1600	1600
TA23	20 × 20	1472	1557	1570	1562.6	1557	1558	1557	1557	1560
TA24	20 × 20	1602	1646	1654	1650.6	1646	1653	1646	1647	1646
TA25	20 × 20	1504	1595	1611	1602	1595	1596	1595	1595	1597
TA26	20 × 20	1539	1645	1658	1652.3	**1643**	1647	1647	1645	1647
TA27	20 × 20	1616	1680	1689	1685.6	1680	1685	1686	1680	1680
TA28	20 × 20	1591	1603	1617	1611.7	1603	1614	1613	1614	1603
TA29	20 × 20	1514	1625	1629	1627.4	1625	1625	1625		1627
TA30	20 × 20	1473	1584	1598	1588.5	1584	1584	1584	1584	1584
TA31	30 × 15	1764	1764	1766	1764.4	1764	1764	1766		1764
TA32	30 × 15	1774	1790	1801	1794.1	**1785**	1793	1790		1795
TA33	30 × 15	1778	1791	1799	1793.7	1791	1799	1791	1793	1796
TA34	30 × 15	1828	1829	1834	1832.1	1829	1832	1832	1829	1831
TA35	30 × 15	2007	2007	2007	2007	2007	2007	2007		2007
TA36	30 × 15	1819	1819	1827	1822.9	1819	1819	1819		1819
TA37	30 × 15	1771	1771	1784	1777.8	1771	1779	1784	1778	1778
TA38	30 × 15	1673	1673	1681	1676.7	1673	1673	1673		1673
TA39	30 × 15	1795	1795	1806	1801.6	1795	1795	1795		1795
TA40	30 × 15	1631	1673	1689	1678.1	**1669**	1680	1979	1674	1676
TA41	30 × 20	1859	2006	2027	2018.7	2008	2022	2022		2018
TA42	30 × 20	1867	1945	1957	1949.3	**1937**	1956	1953	1956	1953
TA43	30 × 20	1809	1848	1874	1863.1	1852	1870	1869	1859	1858
TA44	30 × 20	1927	1983	2003	1992.4	1983	1991	1992	1984	1983
TA45	30 × 20	1997	2000	2000	2000	2000	2004	2000	2000	2000
TA46	30 × 20	1940	2008	2023	2015.5	**2004**	2011	2011	2021	2010
TA47	30 × 20	1789	1897	1908	1902.1	**1894**	1903	1902	1903	1903
TA48	30 × 20	1912	1945	1973	1959.2	**1943**	1962	1962	1953	1955
TA49	30 × 20	1915	1966	1983	1972.6	**1964**	1969	1974		1967
TA50	30 × 20	1807	1925	1932	1927	1925	1931	1927	1928	1931
						%ARE =	0.194	0.518	0.194	0.119
					BRKGA-JSP	%ARE =	-0.023	-0.023	-0.043	-0.023

Newly found upper bounds by BRKGA-JSP are in **bold**.

TABLE 14. Makespan and average percent deviations from best upper bound for problem class DMU (DMU01–DMU40).

				BRKGA-JSP			TS	GES	i-TSAB	AlgFix
Prob	$n \times m$	LB	UB	max	avg	min	min	min	min	min
DMU01	20 × 15	2501	2563	2563	2563	2563	2566	2566	2571	2563
DMU02	20 × 15	2651	2706	2716	2714.5	2706	2711	2706	2715	2706
DMU03	20 × 15	2731	2731	2741	2736.5	2731		2731		2731
DMU04	20 × 15	2601	2669	2679	2672.4	2669		2669		2669
DMU05	20 × 15	2749	2749	2771	2755.4	2749		2749		2749
DMU06	20 × 20	2834	3244	3250	3246.6	3244	3254	3250	3265	3244
DMU07	20 × 20	2677	3046	3063	3058.6	3046		3053		3046
DMU08	20 × 20	2901	3188	3191	3188.3	3188	3191	3197	3199	3188
DMU09	20 × 20	2739	3092	3095	3094.4	3092		3092	3094	3096
DMU10	20 × 20	2716	2984	2985	2984.8	2984		2984	2985	2984
DMU11	30 × 15	3395	3453	3449	3445.8	**3445**	3455	3453	3470	3455
DMU12	30 × 15	3481	3516	3529	3518.9	**3513**	3516	3518	3519	3522
DMU13	30 × 15	3681	3681	3698	3690.6	3681	3681	3697	3698	3687
DMU14	30 × 15	3394	3394	3394	3394	3394		3394	3394	3394
DMU15	30 × 15	3332	3343	3343	3343	3343		3343		3343
DMU16	30 × 20	3726	3759	3769	3758.9	**3751**	3759	3781	3787	3772
DMU17	30 × 20	3697	3836	3870	3850.6	**3830**	3842	3848	3854	3836
DMU18	30 × 20	3844	3846	3847	3845.4	**3844**	3846	3849	3854	3852
DMU19	30 × 20	3650	3775	3803	3791.8	**3770**	3784	3807	3823	3775
DMU20	30 × 20	3604	3712	3718	3715.3	3712	3716	3739	3740	3712
DMU21	40 × 15	4380	4380	4380	4380	4380		4380		4380
DMU22	40 × 15	4725	4725	4725	4725	4725		4725		4725
DMU23	40 × 15	4668	4668	4668	4668	4668		4668		4668
DMU24	40 × 15	4648	4648	4648	4648	4648		4648		4648
DMU25	40 × 15	4164	4164	4164	4164	4164		4164		4164
DMU26	40 × 20	4647	4647	4686	4658.4	4647	4647	4667	4679	4688
DMU27	40 × 20	4848	4848	4848	4848	4848		4848	4848	4848
DMU28	40 × 20	4692	4692	4692	4692	4692		4692		4692
DMU29	40 × 20	4691	4691	4691	4691	4691		4691	4691	4691
DMU30	40 × 20	4732	4732	4732	4732	4732		4732	4732	4749
DMU31	50 × 15	5640	5640	5640	5640	5640		5640		5640
DMU32	50 × 15	5927	5927	5927	5927	5927		5927		5927
DMU33	50 × 15	5728	5728	5728	5728	5728		5728		5728
DMU34	50 × 15	5385	5385	5385	5385	5385		5385		5385
DMU35	50 × 15	5635	5635	5635	5635	5635		5635		5635
DMU36	50 × 20	5621	5621	5621	5621	5621		5621		5621
DMU37	50 × 20	5851	5851	5851	5851	5851		5851	5851	5851
DMU38	50 × 20	5713	5713	5713	5713	5713		5713		5713
DMU39	50 × 20	5747	5747	5747	5747	5747		5747		5747
DMU40	50 × 20	5577	5577	5577	5577	5577		5577		5577

Newly found upper bounds by BRKGA-JSP are in **bold**.

TABLE 15. Makespan and average percent deviations from best upper bound for problem class DMU (DMU41–DMU80).

				BRKGA-JSP			TS	GES	i-TSAB	AlgFix
Prob	$n \times m$	LB	UB	max	avg	min	min	min	min	min
DMU41	20 × 15	2839	3264	3304	3281.9	**3261**		3267	3277	3278
DMU42	20 × 15	3066	3401	3429	3403.9	**3395**	3416	3401	3448	3412
DMU43	20 × 15	3121	3443	3468	3452.7	**3441**	3459	3443	3473	3450
DMU44	20 × 15	3112	3489	3539	3510.7	**3488**	3524	3489	3528	3489
DMU45	20 × 15	2930	3273	3316	3287.3	**3272**	3296	3273	3321	3273
DMU46	20 × 20	3425	4043	4071	4043.2	**4035**	4080	4099	4101	4071
DMU47	20 × 20	3353	3950	3991	3968	**3939**		3972	3973	3950
DMU48	20 × 20	3317	3795	3812	3800.9	**3781**	3795	3810	3838	3813
DMU49	20 × 20	3369	3724	3735	3729.6	**3723**	3735	3754	3780	3725
DMU50	20 × 20	3379	3737	3776	3746.5	**3732**	3761	3768	3794	3742
DMU51	30 × 15	3839	4202	4258	4222.9	**4201**	4218	4247	4260	4202
DMU52	30 × 15	4012	4353	4366	4352.3	**4341**	4362	4380	4383	4353
DMU53	30 × 15	4108	4419	4438	4420.2	**4415**	4428	4450	4470	4419
DMU54	30 × 15	4165	4405	4409	4402.7	**4396**	4405	4424	4425	4413
DMU55	30 × 15	4099	4303	4310	4299.4	**4290**	4308	4331	4332	4321
DMU56	30 × 20	4366	4985	5026	4768.4	**4961**	5025	5051	5079	4985
DMU57	30 × 20	4182	4698	4716	4704.9	4698	4698	4779	4785	4709
DMU58	30 × 20	4214	4787	4759	4752.8	**4751**	4796	4829	4834	4787
DMU59	30 × 20	4199	4638	4641	4633.3	**4630**	4667	4694	4696	4638
DMU60	30 × 20	4259	4805	4786	4777	**4774**	4805	4888	4904	4827
DMU61	40 × 15	4886	5228	5248	5233.3	**5224**	5228	5293	5294	5310
DMU62	40 × 15	5004	5311	5316	5304.4	**5301**	5311	5354	5354	5330
DMU63	40 × 15	5049	5371	5399	5386.6	**5357**	5371	5439	5446	5431
DMU64	40 × 15	5130	5330	5340	5321.8	**5312**	5330	5388	5443	5385
DMU65	40 × 15	5072	5201	5247	5211.5	**5197**	5201	5269	5271	5322
DMU66	40 × 20	5357	5797	5827	5806.6	**5796**	5797	5902	5911	5886
DMU67	40 × 20	5484	5872	5900	5881.3	**5863**	5872	6012	6016	5938
DMU68	40 × 20	5423	5834	5857	5843.7	**5826**	5834	5934	5936	5840
DMU69	40 × 20	5419	5794	5856	5804	**5776**	5794	6002	5891	5868
DMU70	40 × 20	5492	5954	5984	5968.2	**5951**	5954	6072	6096	6028
DMU71	50 × 15	6050	6278	9298	6603.8	6293	6278	6333	6359	6437
DMU72	50 × 15	6223	6520	6593	6560.7	**6503**	6520	6589	6586	6604
DMU73	50 × 15	5935	6249	6297	6250.5	**6219**	6249	6291	6330	6343
DMU74	50 × 15	6015	6316	6354	6312.6	**6277**	6316	6376	6383	6467
DMU75	50 × 15	6010	6236	6326	6282.4	6248	6236	6380	6437	6397
DMU76	50 × 20	6329	6893	6910	6885.4	**6876**	6893	6974	7082	6975
DMU77	50 × 20	6399	6868	6934	6892.7	**6857**	6868	7006	6930	6949
DMU78	50 × 20	6508	6846	6875	6855.7	**6831**	6846	6988	7027	6928
DMU79	50 × 20	6593	7055	7084	7060.9	**7049**	7055	7158	7253	7083
DMU80	50 × 20	6435	6719	6810	6757.9	6736	6719	6843	6998	6861
						% ARE =	0.162	0.629	1.150	0.424
					BRKGA-JSP	%ARE =	-0.155	-0.104	-0.138	-0.104

Newly found upper bounds by BRKGA-JSP are in **bold**.

Acknowledgments

This work has been supported by funds granted by the ERDF through the Programme COMPETE and by the Portuguese Government through FTC, the Foundation for Science and Technology, project PTDC/EGE-GES/117692/2010.

References

Aarts, E., Van Laarhoven, P., Lenstra, J., Ulder, N., 1994. A computational study of local search algorithms for job shop scheduling. INFORMS Journal on Computing 6, 118.

Adams, J., Balas, E., Zawack, D., 1988. The shifting bottleneck procedure for job shop scheduling. Management Science 34, 391–401.

Aiex, R., Binato, S., Resende, M., 2003. Parallel GRASP with path-relinking for job shop scheduling. Parallel Computing 29, 393–430.

Akers, S., 1956. A graphical approach to production scheduling problems. Operations Research 4, 244–245.

Applegate, D., Cook, W., 1991. A computational study of the job-shop scheduling problem. ORSA Journal on Computing 3, 149–156.

Baker, J., McMahon, G., 1985. Scheduling the general job-shop. Management Science 31, 594–598.

Balas, E., Vazacopoulos, A., 1998. Guided local search with shifting bottleneck for job shop scheduling. Management Science , 262–275.

Bean, J.C., 1994. Genetics and random keys for sequencing and optimization. ORSA Journal on Computing 6, 154–160.

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

Binato, S., Hery, W., Loewenstern, D., Resende, M., 2002. A GRASP for job shop scheduling, in: Ribeiro, C., Hansen, P. (Eds.), Essays and surveys in metaheuristics, pp. 58–79.

Blazewicz, J., Domschke, W., Pesch, E., 1996. The job shop scheduling problem: Conventional and new solution techniques. European journal of operational research 93, 1–33.

Brinkkötter, W., Brucker, P., 2001. Solving open benchmark instances for the job-shop problem by parallel head-tail adjustments. Journal of Scheduling 4, 53–64.

Brucker, P., 1988. An efficient algorithm for the job-shop problem with two jobs. Computing 40, 353–359.

Brucker, P., Jurisch, B., Sievers, B., 1994. A branch and bound algorithm for the job-shop scheduling problem* 1. Discrete Applied Mathematics 49, 107–127.

Carlier, J., Pinson, E., 1989. An algorithm for solving the job-shop problem. Management science , 164–176.

Carlier, J., Pinson, E., 1990. A practical use of Jackson's preemptive schedule for solving the job-shop problem. Annals of Operations Research 26, 269–287.

Cheng, R., Gen, M., Tsujimura, Y., 1996. A tutorial survey of job-shop scheduling problems using genetic algorithms - I. Representation : Genetic algorithms and industrial engineering. Computers and Industrial Engineering 30, 983–997.

Cheng, R., Gen, M., Tsujimura, Y., 1999. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: Hybrid genetic search strategies. Computers & Industrial Engineering 36, 343–364.

Davis, L., 1985. Job shop scheduling with genetic algorithms, in: Proceedings of the 1st International Conference on Genetic Algorithms, L. Erlbaum Associates Inc.. p. 140.

DeJong, K., Spears, W., 1991. On the virtues of parameterised uniform crossover, in: Belew, R.K., Booker, L.B. (Eds.), Proceedings of the Fourth International Conference on Genetic Algorithms. Morgan Kaufman, San Mateo, CA, pp. 230–236.

Della Croce, F., Tadei, R., Volta, G., 1995. A genetic algorithm for the job shop problem. Computers & Operations Research 22, 15–24.

Demirkol, E., Mehta, S., Uzsoy, R., 1997. A computational study of shifting bottleneck procedures for shop scheduling problems. Journal of Heuristics 3, 111–137.

Dorndorf, U., Pesch, E., 1995. Evolution based learning in a job shop scheduling environment. Computers & Operations Research 22, 25–40.

Fisher, H., Thompson, G., 1963. Probabilistic learning combinations of local job-shop scheduling rules, in: Muth, J., Thompson, G. (Eds.), Industrial Scheduling. Prentice Hall, Englewood Cliffs, New Jersey, pp. 225–251.

French, S., 1982. Sequencing and scheduling: An introduction to the mathematics of the job-shop. Halsted Press.

Gen, M., Tsujimura, Y., Kubota, E., 1994. Solving job-shop scheduling problems by genetic algorithm, in: IEEE International Conference on Systems, Man, and Cybernetics. "Humans, Information and Technology", pp. 1577–1582.

Giffler, B., Thompson, G., 1960. Algorithms for solving production-scheduling problems. Operations Research , 487–503.

Gonçalves, J.F., Resende, M.G.C., Mendes, J.J.M., 2010. A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. Journal of Heuristics , 1–20.

Gonçalves, J., Resende, M., 2011. A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. J. of Combinatorial Optimization 22, 180–201.

Gonçalves, J.F., Mendes, J.J.M., Resende, M.G.C., 2005. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 167, 77–95.

Gonçalves, J.F., Resende, M.G.C., 2011. Biased random-key genetic algorithms for combinatorial optimization. Journal of Heuristics 17, 487–525.

Gray, C., Hoesada, M., 1991. Matching heuristic scheduling rules for job shops to the business sales level. Production and Inventory Management Journal 4, 12–17.

Henning, A., 2002. Practical job shop scheduling problems (in german). Ph.d. thesis,. Friedrich-Schiller-University Jena, Jena, Germany.

Jain, A., 1998. A Multi-Level Hybrid Framework for the Deterministic Job-Shop Scheduling Problem. Ph.d. thesis,. Department of Applied Physics and Electrical and Mechanical Engineering, University of Dundee, Dundee, Scottland, UK .

Jain, A., Meeran, S., 1999. Deterministic job-shop scheduling: Past, present and future. European Journal of Operational Research 113, 390–434.

Lageweg, B., Lenstra, J., Rinnooy Kan, A., 1977. Job-shop scheduling by implicit enumeration. Management Science 24, 441–450.

Lawrence, S., 1984. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement). Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania .

Lenstra, J., Rinnooy Kan, A., 1979. Computational complexity of discrete optimization problems. Annals of Discrete Mathematics 4, 121–140.

Lourenço, H., 1995. Job-shop scheduling: Computational study of local search and large-step optimization methods. European Journal of Operational Research 83, 347–364.

Lourenço, H., Zwijnenburg, M., 1996. Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem, in: Osman, I., Kelly, J. (Eds.), Meta-Heuristics: Theory & Applications. Kluwer Academic Publishers, Boston, pp. 219–236.

Mendes, J., Gonçalves, J., Resende, M., 2009. A random key based genetic algorithm for the resource constrained project scheduling problem. Computers & Operations Research 36, 92–109.

Nowicki, E., Smutnicki, C., 1996. A fast taboo search algorithm for the job shop problem. Management Science 42, 797–813.

Nowicki, E., Smutnicki, C., 2002. Some new tools to solve the job shop problem. Technical Report 60/02. Institute of Engineering Cybernetics, Wroclaw University of Technology.

Nowicki, E., Smutnicki, C., 2005. An advanced tabu search algorithm for the job shop problem. Journal of Scheduling 8, 145–159.

Pardalos, P., Shylo, O., 2006. An algorithm for the job shop scheduling problem based on global equilibrium search techniques. Computational Management Science 3, 331–348.

Pardalos, P., Shylo, O., Vazacopoulos, A., 2010. Solving job shop scheduling problems utilizing the properties of backbone and "big valley". Computational Optimization and Applications 47, 61–76.

Pinson, E., 1995. The job shop scheduling problem: A concise survey and some recent developments, in: Chrétienne, P., Coffman, E., Lenstra, J.K., Liu, Z. (Eds.), Scheduling Theory and Its Applications. John Wiley and Sons, pp. 277–293.

Resende, M., Toso, R., Gonçalves, J., Silva, R., 2012. A biased random-key genetic algorithm for the steiner triple covering problem. Optimization Letters 6, 605–619.

Sabuncuoglu, I., Bayiz, M., 1999. Job shop scheduling with beam search. European Journal of Operational Research 118, 390–412.

Schilham., R., 2001. Commonalities in local search. Phd thesis,. Utrecht University, The Netherlands.

Schrage, L., 1970. Solving resource-constrained network problems by implicit enumeration - non-preemptive case. Operations Research 18, 263–278.

Shi, G., Iima, H., Sannomiya, N., 1996. A new encoding scheme for solving job shop problems by genetic algorithm, in: Proceedings of the 35th IEEE Decision and Control, pp. 4395–4400.

Storer, R., Wu, S., Vaccari, R., 1992. New search spaces for sequencing problems with application to job shop scheduling. Management Science 38, 1495–1509.

Taillard, E., 1994. Parallel taboo search techniques for the job shop scheduling problem. ORSA journal on Computing 6, 108–108.

Vaessens, R., Aarts, E., Lenstra, J., 1996. Job shop scheduling by local search. INFORMS Journal on Computing 8, 302–317.

Van Laarhoven, P., Aarts, E., Lenstra, J., 1992. Job shop scheduling by simulated annealing. Operations Research 40, 113–125.

Wennink, M., 1995. Algorithm support for automated planning boards. Ph.d. thesis,. Department of Mathematics and Computing Science, Eindhoven University of Technology,.

Williamson, D., Hall, L., Hoogeveen, J., Hurkens, C., Lenstra, J., Sevast'janov, S., Shmoys, D., 1997. Short shop schedules. Operations Research 45, 288–294.

Yamada, T., Nakano, R., 1992. A genetic algorithm applicable to large-scale job-shop problems, in: Proceedings of 2nd International Workshop on Parallel Problem Solving from Nature, pp. 281–290.

Zhang, C., Li, P., Guan, Z., Rao, Y., 2007. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. Computers & Operations Research 34, 3229–3242.

Zhang, C., Li, P., Rao, Y., Guan, Z., 2008. A very fast TS/SA algorithm for the job shop scheduling problem. Computers & Operations Research 35, 282–294.

LIAAD, INESCTEC, Faculdade de Economia, Universidade do Porto,, Rua Dr. Roberto Frias, s/n, 4200-464 Porto, Portugal

E-mail address: jfgoncal@fep.up.pt

Algorithms and Optimization Research Department, AT&T Labs Research,, 180 Park Avenue, Room C241, Florham Park, NJ 07932 USA

E-mail address: mgcr@research.att.com