Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 167 (2005) 77–95
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Discrete Optimization

A hybrid genetic algorithm for the job shop scheduling problem

José Fernando Gonçalves [a], Jorge José de Magalhães Mendes [b], Maurício G.C. Resende [c,*]

[a] *Faculdade de Economia da Universidade do Porto, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal*
[b] *Depto. de Engenharia Informática, Instituto Superior de Engenharia do Porto, Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto, Portugal*
[c] *Internet and Network Systems Research, AT&T Labs Research, Florham Park, NJ 07932, USA*

Received 17 February 2003; accepted 10 March 2004
Available online 24 June 2004

Abstract

This paper presents a hybrid genetic algorithm for the job shop scheduling problem. The chromosome representation of the problem is based on random keys. The schedules are constructed using a priority rule in which the priorities are defined by the genetic algorithm. Schedules are constructed using a procedure that generates parameterized active schedules. After a schedule is obtained a local search heuristic is applied to improve the solution. The approach is tested on a set of standard instances taken from the literature and compared with other approaches. The computation results validate the effectiveness of the proposed algorithm.

Keywords: Job shop; Scheduling; Genetic algorithm; Heuristics; Random keys

1. Introduction

The job shop scheduling problem (JSP), may be described as follows: given n jobs, each composed of several operations that must be processed on m machines. Each operation uses one of the m machines for a fixed duration. 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. The operations of a given job have to be processed in a given order. The problem consists in finding a schedule of the operations on the machines, taking into account the precedence constraints, that minimizes the makespan ($C_{\max}$), that is, the finish time of the last operation completed in the schedule.

* Corresponding author. Tel.: +1-973-3608444; fax: +1-973-3608178.
E-mail addresses: jfgoncal@fep.up.pt (J.F. Gonçalves), jjm@isep.ipp.pt (J.J. de Magalhães Mendes), mgcr@research.att.com (M.G.C. Resende).

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2004.03.012

Let $J = \{0, 1, \ldots, n, n+1\}$ denote the set of operations to be scheduled and $M = \{1, \ldots, m\}$ the set of machines. The operations 0 and $n+1$ are dummy, have no duration and represent the initial and final operations. The operations are interrelated by two kinds of constraints. First, the precedence constraints, which force each operation j to be scheduled after all predecessor operations, P_j are completed. Second, operation j can only be scheduled if the machine it requires is idle. Further, let d_j denote the (fixed) duration (processing time) of operation j.

Let F_j represent the finish time of operation j. A schedule can be represented by a vector of finish times $(F_1, F_m, \ldots, F_{n+1})$. Let $A(t)$ be the set of operations being processed at time t, and let $r_{j,m} = 1$ if operation j requires machine m to be processed and $r_{j,m} = 0$ otherwise.

The conceptual model of the JSP can be described the following way:

$$\text{Minimize} \quad F_{n+1}(C_{\max}) \tag{1}$$

$$\text{subject to :} \tag{2}$$

$$F_k \leqslant F_j - d_j, \quad j = 1, \ldots, n+1; \;\; k \in P_j, \tag{3}$$

$$\sum_{j \in A(t)} r_{j,m} \leqslant 1, \quad m \in M; \;\; t \geqslant 0, \tag{4}$$

$$F_j \geqslant 0, \quad j = 1, \ldots, n+1. \tag{5}$$

The objective function (1) minimizes the finish time of operation $n+1$ (the last operation), and therefore minimizes the *makespan*. Constraints (3) impose the precedence relations between operations and constraints (4) state that one machine can only process one operation at a time. Finally (5) forces the finish times to be non-negative.

The JSP is amongst the hardest combinatorial optimization problems. The JSP is NP-hard (Lenstra and Rinnooy Kan, 1979), and has also proven to be computationally challenging.

Historically JSP has been primarily treated using the following approaches:

- *Exact methods*: Giffler and Thompson (1960), Brucker et al. (1994) and Williamson et al. (1997);
- *Branch and bound*: Lageweg et al. (1977), Carlier and Pinson (1989, 1990), Applegate and Cook (1991) and Sabuncuoglu and Bayiz (1999). Carlier and Pinson (1989) have been successful in solving the notorious 10×10 instance of Fisher and Thompson proposed in 1963 and only solved 20 years later;
- *Heuristic procedures based on priority rules*: French (1982), Gray and Hoesada (1991) and Gonçalves and Mendes (1994);
- *Shifting bottleneck*: Adams et al. (1988).

Problems of dimension 15×15 are still considered to be beyond the reach of today's exact methods. Over the last decade, a growing number of metaheuristic procedures have been presented to solve hard optimization problems:

- *Simulated annealing*: Laarhoven et al. (1992) and Lourenço (1995);
- *Tabu search*: Taillard (1994), Lourenço and Zwijnenburg (1996) and Nowicki and Smutnicki (1996);
- *Genetic algorithms*: Davis (1985), Storer et al. (1992), Aarts et al. (1994), Croce et al. (1995), Dorndorf and Pesch (1995), Gonçalves and Beirão (1999) and Oliveira (2000).

Additionally, some researchers have developed local search procedures: Lourenço (1995), Vaessens et al. (1996), Lourenço and Zwijnenburg (1996) and Nowicki and Smutnicki (1996). Surveys of heuristic methods for the JSP are given in Pinson (1995), Vaessens et al. (1996) and Cheng et al. (1999).

A comprehensive survey of job shop scheduling techniques can be found in Jain and Meeran (1999). Recently Wang and Zheng (2001) developed a hybrid optimization strategy for JSP, Binato et al. (2002) present a greedy randomized adaptive search procedure (GRASP) for JSP and Aiex et al. (2003) introduced a parallel GRASP with path-relinking for JSP.

In this paper, we present a new hybrid genetic algorithm for the job shop scheduling problem. The remainder of the paper is organized as follows. In Section 2, we present the different classes of schedules. In Section 3, we present our approach to solve the job shop scheduling problem: genetic algorithm, schedule generation procedure, and local search procedure. Section 4 reports the computational results and the conclusions are made in Section 5.

2. Parameterized active schedules

The optimal schedule is in the set of all active schedules. However, the set of active schedules is usually very large and contains many schedules with relatively large delay times, and therefore with poor quality in terms of makespan. In order to reduce the solution space we used the concept of parameterized active schedules (Gonçalves and Beirão, 1999).

The basic idea of parameterized active schedules consists in controlling the delay times that each operation is allowed. By controlling the maximum delay time allowed, one can reduce or increase the solution space. A maximum delay time equal to zero is equivalent to restricting the solution space to non-delay schedules and a maximum delay time equal to infinity is equivalent to allow active schedules.

Fig. 1 illustrates where the set of parameterized active schedules is located relative to the class of semi-active, active, and non-delay schedules.

Section 3.3 presents a detailed pseudo-code procedure to generate parameterized active schedules.

3. New approach

The new approach combines a genetic algorithm, a schedule generator procedure that generates parameterized active schedules and a local search procedure.

The genetic algorithm is responsible for evolving the chromosomes which represent the priorities of the operations and delay times. For each chromosome the following three phases are applied:

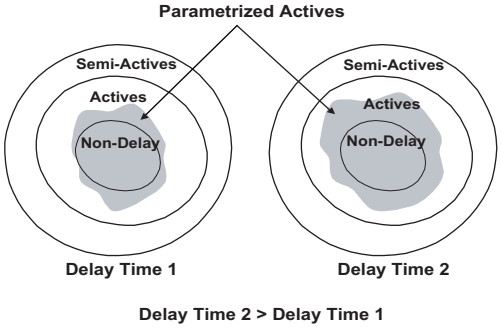

Fig. 1. Parameterized active schedules.

- *Decoding of priorities and delay times*. This phase is responsible for transforming the chromosome supplied by the genetic algorithm into the priorities of the operations and the delay times.
- *Schedule generation*. This phase makes use of the priorities and the delay times defined in the first phase and constructs parameterized active schedules.
- *Schedule improvement*. This phase makes use of a local search procedure to improve the solution obtained in the schedule generation phase.

After a schedule is obtained the corresponding quality (makespan, smaller is better) is feedback to the genetic algorithm. Fig. 2 illustrates the sequence of steps applied to each chromosome generated by the genetic algorithm.

Details about each of these phases will be presented in the next sections.

3.1. Genetic algorithm

Genetic algorithms are adaptive methods, which may be used to solve search and optimization problems (Beasley et al., 1993). They are based on the genetic process of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection, i.e. *survival of the fittest*, first clearly stated by Charles Darwin in *The Origin of Species*. By mimicking this process, genetic algorithms are able to *evolve* solutions to real world problems, if they have been suitably encoded.

Before a genetic algorithm can be run, a suitable *encoding* (or *representation*) for the problem must be devised. A *fitness function* is also required, which assigns a figure of merit to each encoded solution. During the run, parents must be *selected* for reproduction, and *recombined* to generate offspring (see Fig. 3).

It is assumed that a potential solution to a problem may be represented as a set of parameters. These parameters (known as *genes*) are joined together to form a string of values (*chromosome*). In genetic ter-

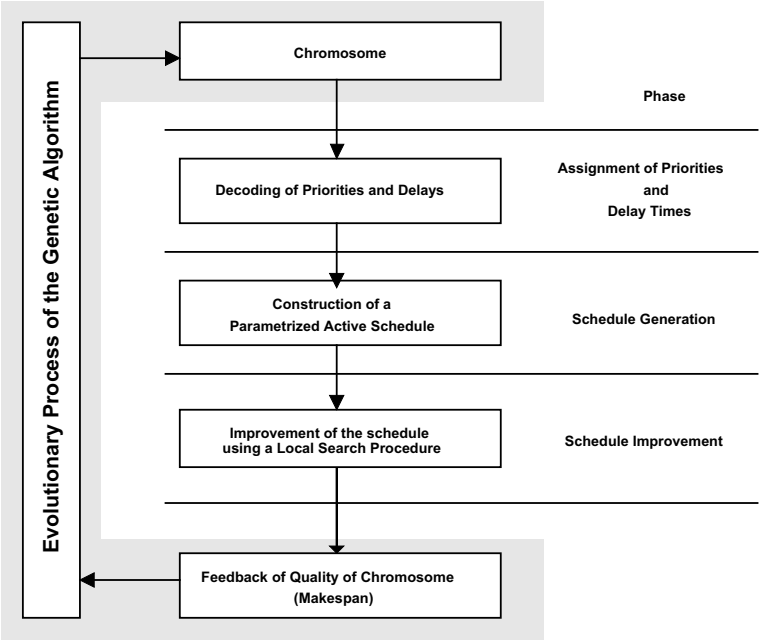

Fig. 2. Architecture of the new approach.

```
Genetic Algorithm
{
    Generate initial population P_t
    Evaluate population P_t
    While stopping criteria not satisfied Repeat
    {
        Select elements from P_t to copy into P_t+1
        Crossover elements of P_t and put into P_t+1
        Mutation elements of P_t and put into P_t+1
        Evaluate new population P_t+1
        P_t = P_t+1
    }
}
```

Fig. 3. A standard genetic algorithm.

minology, the set of parameters represented by a particular chromosome is referred to as an *individual*. The fitness of an individual depends on its chromosome and is evaluated by the fitness function.

The individuals, during the reproductive phase, are selected from the population and *recombined*, producing offspring, which comprise the next generation. Parents are randomly selected from the population using a scheme, which favors fitter individuals. Having selected two parents, their chromosomes are *recombined*, typically using mechanisms of *crossover* and *mutation*. Mutation is usually applied to some individuals, to guarantee population diversity.

3.1.1. Chromosome representation and decoding

The genetic algorithm described in this paper uses a random key alphabet $\mathbf{U}(0,1)$ and an evolutionary strategy identical to the one proposed by Bean (1994). The important feature of random keys is that all offspring formed by crossover are feasible solutions. This is accomplished by moving much of the feasibility issue into the objective function evaluation. If any random key vector can be interpreted as a feasible solution, then any crossover vector is also feasible. Through the dynamics of the genetic algorithm, the system learns the relationship between random key vectors and solutions with good objective function values.

A chromosome represents a solution to the problem and is encoded as a vector of random keys (random numbers). Each solution chromosome is made of $\boldsymbol{2n}$ genes where $\boldsymbol{n}$ is the number of operations.

$$\boldsymbol{Chromosome} = (\boldsymbol{gene}_1, \boldsymbol{gene}_2, \ldots, \boldsymbol{gene}_n, \boldsymbol{gene}_{n+1}, \ldots, \boldsymbol{gene}_{2n})$$

The first $\boldsymbol{n}$ genes are used as operations priorities, i.e.

$$\boldsymbol{Priority}_j = \boldsymbol{Gene}_j.$$

The genes between $\boldsymbol{n}+\mathbf{1}$ and $\boldsymbol{2n}$ are used to determine the delay times used when scheduling an operation. The delay time used by each scheduling iteration $\boldsymbol{g}$, $\boldsymbol{Delay}_g$, is calculated by the following expression:

$$\boldsymbol{Delay}_g = \boldsymbol{gene}_g \times \mathbf{1.5} \times \boldsymbol{MaxDur},$$

where $\boldsymbol{MaxDur}$ is the maximum duration of all operations. The factor 1.5 was obtained after experimenting with values between 1.0 and 2.0 in increments of 0.1. The value giving the best results was chosen.

3.1.2. Evolutionary strategy

To breed good solutions, the random key vector population is operated upon by a genetic algorithm. There are many variations of genetic algorithms obtained by altering the reproduction, crossover, and mutation operators. The reproduction and crossover operators determine which parents will have offspring, and how genetic material is exchanged between the parents to create those offspring. Mutation allows for

random alteration of genetic material. Reproduction and crossover operators tend to increase the quality of the populations and force convergence. Mutation opposes convergence and replaces genetic material lost during reproduction and crossover.

Reproduction is accomplished by first copying some of the best individuals from one generation to the next, in what is called an elitist strategy (Goldberg, 1989). The advantage of an elitist strategy over traditional probabilistic reproduction is that the best solution is monotonically improving from one generation to the next. The potential downside is population convergence to a local minimum. This can, however, be overcome by high mutation rates as described below.

Parameterized uniform crossovers (Spears and Dejong, 1991) are employed in place of the traditional one-point or two-point crossover. After two parents are chosen randomly from the full, old population (including chromosomes copied to the next generation in the elitist pass), at each gene we toss a biased coin to select which parent will contribute the allele. Fig. 4 presents an example of the crossover operator. It assumes that a coin toss of heads selects the gene from the first parent, a tails chooses the gene from the second parent, and that the probability of tossing a heads is for example 0.7 (this value is determined empirically). Fig. 4 shows one potential crossover outcome:

Rather than the traditional gene-by-gene mutation with very small probability at each generation, one or more new members of the population are randomly generated from the same distribution as the original population. This process prevents premature convergence of the population, like in a mutation operator, and leads to a simple statement of convergence.

Fig. 5 depicts the transitional process between two consecutive generations.

Coin toss	*H*	*H*	*T*	*H*	*T*
Parent 1	*0.57*	*0.93*	*0.36*	*0.12*	*0.78*
Parent 2	*0.46*	*0.35*	*0.59*	*0.89*	*0.23*
Offspring	*0.57*	*0.93*	*0.59*	*0.12*	*0.23*

Fig. 4. Example of parameterized uniform crossover.

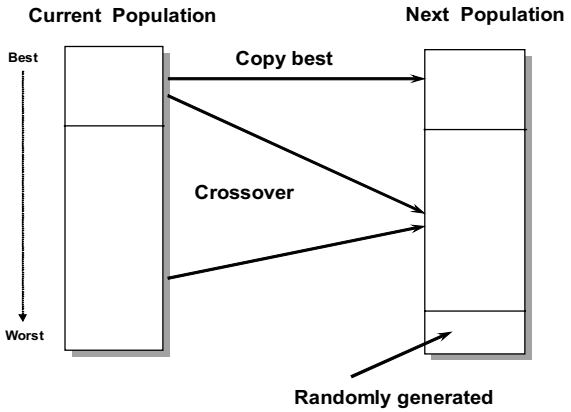

Fig. 5. Transitional process between consecutive generations.

3.2. Schedule generation procedure

The procedure used to construct parameterized active schedules is based on a scheduling generation scheme that does time incrementing. For each iteration g, there is a scheduling time t_g. The active set comprises all operations which are active at t_g, i.e. $A_g = \{j, \in J | F_j - d_j \leqslant t_g < F_j\}$. The remaining machine capacity at t_g is given by $RMC_m(t_g) = 1 - \sum_{j \in A_g} r_{j,m}$. S_g comprises all operations which have been scheduled up to iteration g, and F_g comprises the finish times of the operations in S_g. Let $Delay_g$ be the delay time associated with iteration g, and let E_g comprise all operations which are precedence feasible in the interval $[t_g, t_g + Delay_g]$, i.e.

$$E_g = \{j \in J \setminus S_{g-1} | F_i \leqslant t_g + Delay_g (i \in P_j)\}.$$

The algorithmic description of the scheduling generation scheme used to generate parameterized active schedules is given by pseudo-code shown in Fig. 6.

The makespan of the solution is given by the maximum finish time of all predecessors operations of operation $n+1$, i.e. $F_{n+1} = Max_{l \in P_{n+1}} \{F_l\}$.

The basic idea of parametrized active schedules is incorporated in the selection step of the procedure,

$$j^* = \text{argmax}_{j \in E_g} \{PRIORITY_j\}.$$

The set E_g is responsible for forcing the selection to be made only amongst operations which will not cause a delay smaller or equal to the maximum allowed delay. Fig. 7 illustrates the selection step.

Initialization: $g = 1,\ t_1 = 0,\ A_0 = \{0\},\ F_0 = \{0\},\ S_0 = \{0\},\ RD_m(0) = 1\ \ (m \in M)$

while $|S_g| < n+1$ **repeat**

{

Update E_g

while $E_g \neq \{\}$ **repeat**

{

Select operation with highest priority

$$j^* = \underset{j \in E_t}{\text{argmax}} \{ PRIORITY_j \}$$

Calculate earliest finish time (in terms of precedence only)

$$EF_{j^*} = \max_{i \in P_j} \{F_i\} + d_{j^*}$$

Calculate the earliest finish time (in terms of precedence and capacity)

$$F_{j^*} = \min \left\{ t \in \left[EF_{j^*} - d_{j^*}, \infty \right] \cap F_g \mid r_{j^*,m} \leq RMC_m(\tau), \right.$$

$$\left. r_{j^*,m} > 0,\ \tau \in \left[t, t + d_{j^*} \right] \right\} + d_{j^*}$$

Update $S_g = S_{g-1} \cup \{ j^* \}$, $F_g = F_{g-1} \cup \{ F_{j^*} \}$

Iteration increment: $g = g+1$

Update $A_g, E_g, RD_m(t) \mid t \in \left[F_{j^*} - d_{j^*}, F_{j^*} \right], m \in M \mid r_{j^*,m} > 0$

}

Determine the time associated with iteration g

$$t_g = \min \{ t \in F_{g-1} \mid t > t_{g-1} \}$$

}

Fig. 6. Pseudo-code used to construct parameterized active schedules.

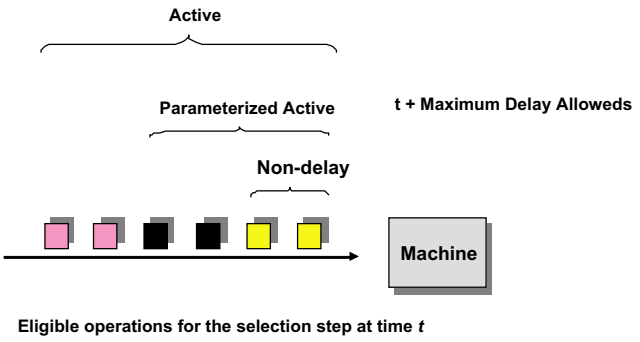

Fig. 7. Eligible operations for different types of schedules.

3.3. Local search procedure

Since there is no guarantee that the schedule obtained in the construction phase is locally optimal with respect to the local neighborhood being adopted, local search may be applied to attempt to decrease the makespan. We employ the two exchange local search, based on the disjunctive graph model of Roy and Sussmann (1964) and the neighborhood of Nowicki and Smutnicki (1996).

The local search procedure begins by identifying the critical path in the solution obtained by the schedule generation procedure. Any operation on the critical path is called a critical operation. It is possible to decompose the critical path into a number of blocks where a block is a maximal sequence of adjacent critical operations that require the same machine.

In this paper, we use the approach of Nowicki and Smutnicki (1996) (see Fig. 8). In this approach, if a job predecessor and a machine predecessor of a critical operation are also critical, then choose the predecessor (from among these two alternatives) which appears first in the operation sequence. The critical path thus gives rise to the following neighborhood of moves. Given b blocks, if $1 < l < b$, then swap only

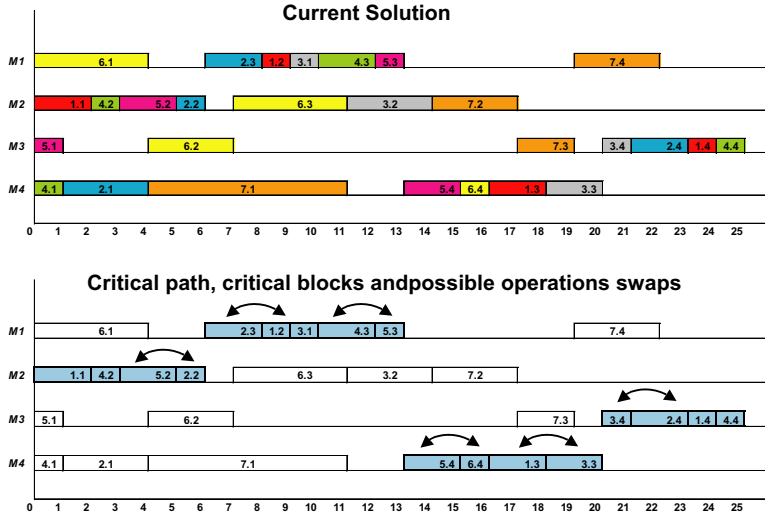

Fig. 8. Neighborhood of Nowicki and Smutnicki (1996).

the last two and first two block operations. Otherwise, if $l = 1$ (b) swap only the last (first) two block operations (see Fig. 8). In the case where the first and/or last block contains only two operations, these operations are swapped. If a block contains only one operation, then no swap is made.

If the swap improves the makespan, it is accepted. Otherwise, the swap is undone. Once a swap is accepted, the critical path may change and a new critical path must be identified. If no swap of first or last operations in any block of critical path improves the makespan, the local search ends.

The algorithmic description of the Local Search Procedure is given in the pseudo-code shown in Fig. 9.

Example

We consider a problem with 2 jobs and 2 operations to be scheduled in 2 machines, Table 1 presents the respective data.

Below we present the details of the approach assuming the genetic algorithm supplied the following chromosome:

$$\textbf{\textit{Chromosome}} = (\mathbf{0.20}, \mathbf{0.22}, \mathbf{0.10}, \mathbf{0.90}, \mathbf{0.14}, \mathbf{0.24}, \mathbf{0.25}, \mathbf{0.70}).$$

Decoding

As mentioned in Section 3.1.1 the first 4 genes are used as operations priorities and last 4 genes are used to determine the delay times for each scheduling iteration g, $\textbf{\textit{Delay}}_g$. Table 2 summarizes the results of the decoding calculations.

```
Local_Search ( CurrentSolution )

    do
    {
        CurrentSolutionUpdated = False

        Determine the critical path and all critical blocks of CurrentSolution

        while Unprocessed blocks and not CurrentSolutionUpdated do
        {
            if not First Critical Block then

                NewSolution := Swap first two operations of block in CurrentSolution

                if Makespan ( NewSolution ) < Makespan ( CurrentSolution ) then
                    CurrentSolution = NewSolution
                    CurrentSolutionUpdated = true
                endif
            endif

            if not Last Critical Block and not CurrentSolutionUpdated then

                NewSolution = Swap last two operations of block in CurrentSolution

                if Makespan ( NewSolution ) < Makespan ( CurrentSolution ) then
                    CurrentSolution = NewSolution
                    CurrentSolutionUpdated = true
                endif
            endif
        }
    }
    until CurrentSolutionUpdated = false

return CurrentSolution
```

Fig. 9. Pseudo-code for the local search procedure.

Table 1
Example data

Operation sequence	Machine	Processing time	Operation number
Job 1			
1st	2	4	1
2nd	1	2	2
Job 2			
1st	1	1	3
2nd	2	3	4

Table 2
Priorities and delay times

Operation/iteration j/g	$PRIORITY_j$	$Delay_g$
1	0.20	0.84
2	0.22	1.44
3	0.25	1.50
4	0.90	4.20

Schedule generation

The schedule is generated using the schedule generation procedure given in Section 3.2 and the priorities and delay times presented in Table 2. The detailed calculations follow.

Initialization

$t_1 = 0, A_0 = \{0\}, F_0 = \{0\}, S_0 = \{0\}$

$RD_m(t) =$

m \ t	1	2	3	4	5	6	7	8	9	10	>10
1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1

Iteration $g = 1$

Update $E_1 = \{j \in J \setminus S_0 | F_i \leqslant 0 + 0.84(i \in P_j)\} = \{1, 3\}$

Select operation with highest priority: $j^* = \text{argmax}_{j \in E_1}\{0.20, 0.25\} = 3$

Calculate earliest finish time (in terms of precedence only): $FMC_3 = 0 + 1 = 1$

Calculate the earliest finish time (in terms of precedence and capacity): $F_3 = 0 + 1 = 1$

Update $S_1 = S_0 \cup \{3\} = \{0\} \cup \{3\} = \{0, 3\}$

Update $F_1 = F_0 \cup \{1\} = \{0\} \cup \{1\} = \{0, 1\}$

Increment iteration counter $g = 1 + 1 = 2$

The partial schedule after iteration $g = 1$ is illustrated in Fig. 10.

Iteration $g = 2$

Update $A_2 = \{3\}$, $E_2 = \{j \in J \setminus S_1 | F_i \leqslant 0 + 1.44(i \in P_j)\} = \{1, 4\}$

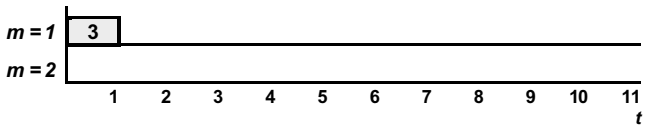

Fig. 10. Partial schedule after operation 3 scheduled.

$RD_m(t) =$

m \ t	1	2	3	4	5	6	7	8	9	10	>10
1	0	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	1

Select operation with highest priority $j^* = \text{argmax}_{j \in E_2}, \{0.20, 0.90\} = 4$
Calculate earliest finish time (in terms of precedence only): $FMC_4 = 1 + 3 = 4$
Calculate the earliest finish time (in terms of precedence and capacity): $F_4 = 1 + 3 = 4$
Update $S_1 = S_1 \cup \{4\} = \{0, 3\} \cup \{4\} = \{0, 3, 4\}$
Update $F_2 = F_1 \cup \{4\} = \{0, 1\} \cup \{4\} = \{0, 1, 4\}$
Increment iteration counter $g = 2 + 1 = 3$
The partial schedule after iteration $g = 2$ is illustrated in Fig. 11.

Iteration $g = 3$
Update $A_3 = \{3\}$, $E_3 = \{j \in J \setminus S_2 | F_i \leqslant 0 + 1.50(i \in P_j)\} = \{1\}$

$RD_m(t) =$

m \ t	1	2	3	4	5	6	7	8	9	10	>10
1	0	1	1	1	1	1	1	1	1	1	1
2	1	0	0	0	1	1	1	1	1	1	1

Select operation with highest priority: $j^* = \text{argmax}_{j \in E_3}, \{0.20, \} = 1$
Calculate the earliest finish time (in terms of precedence only): $FMC_1 = 0 + 4 = 4$
Calculate the earliest finish time (in terms of precedence and capacity): $F_1 = 4 + 4 = 8$
Update $S_3 = S_2 \cup \{1\} = \{0, 3, 4\} \cup \{1\} = \{0, 1, 3, 4\}$
Update $F_3 = F_2 \cup \{8\} = \{0, 1, 4\} \cup \{8\} = \{0, 1, 4, 8\}$
Increment iteration counter $g = 3 + 1 = 4$
The partial schedule after iteration $g = 3$ is illustrated in Fig. 12.

Iteration $g = 4$
Update $A_4 = \{3\}$, $E_4 = \{j \in J \setminus S_3 | F_i \leqslant 0 + 4.20(i \in P_j)\} = \{\ \}$

$RD_m(t) =$

m \ t	1	2	3	4	5	6	7	8	9	10	>10
1	0	1	1	1	1	1	1	1	1	1	1
2	1	0	0	0	0	0	0	0	1	1	1

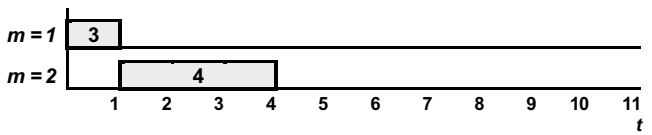

Fig. 11. Partial schedule after operation 4 scheduled.

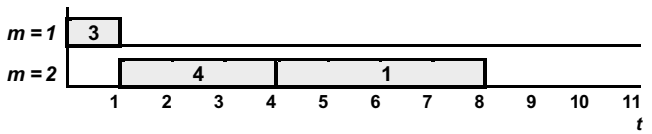

Fig. 12. Partial schedule after operation 13 scheduled.

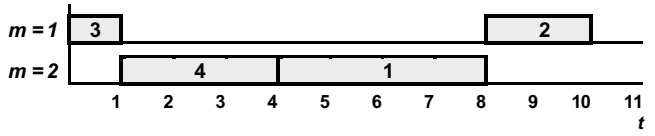

Fig. 13. Partial schedule after operation 2 scheduled.

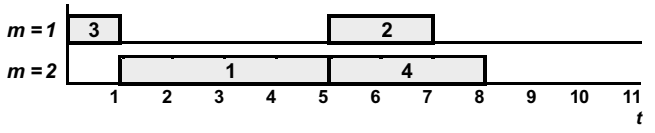

Fig. 14. Schedule after the swap of operations 1 and 4.

Determine the time associated with iteration $g = 4$ until $E_g \neq \{\ \}$
$t_4 = \min\{1, 4, 8\} = 1$
Update $E_4 = \{j \in J \setminus S_3 | F_i \leqslant 1 + 4.20(i \in P_j)\} = \{\ \}$
$t_4 = \min\{4, 8\} = 4$
Update $E_4 = \{j \in J \setminus S_3 | F_i \leqslant 4 + 4.20(i \in P_j)\} = \{2\}$
Select operation with highest priority: $j^* = \underset{j \in E_4}{\operatorname{argmax}}\{0.22\} = \{2\}$
Calculate earliest finish time (in terms of precedence only): $FMC_2 = 8 + 2 = 10$
Calculate the earliest finish time (in terms of precedence and capacity): $F_2 = 8 + 2 = 10$
Update $S_4 = S_3 \cup \{2\} = \{0, 1, 3, 4\} \cup \{2\} = \{0, 1, 3, 4\} \cup \{2\} = \{0, 1, 2, 3, 4\}$
Update $F_4 = F_3 \cup \{10\} = \{0, 1, 4, 8\} \cup \{10\} = \{0, 1, 4, 8, 10\}$
Increment iteration counter $g = 4 + 1 = 5$

The partial schedule after iteration $g = 4$ is illustrated in Fig. 13.
After this iteration the schedule is completed and we can compute the corresponding makespan as

$$Makespan = \max\{8, 10\} = 10.$$

Local search procedure

The local search procedure begins by identifying the critical path in the solution obtained by the schedule generation procedure, Fig. 13. The critical path is defined by operations **3–4–1–2** has only the critical block 4–1 in machine 2. If we swap operations 4 and 1 we obtain a better schedule with a makespan of 8, see Fig. 14.

The critical path corresponding to the schedule in Fig. 14 is defined by operations 3–1–4 and contains only the critical block 1–4 in machine 2. Swapping operations 1 and 4 leads to a schedule with a makespan of 10 which is worse than 8 and so we stop the local search procedure.

4. Computational results

To illustrate the effectiveness of the algorithm described in this paper, we consider 43 instances from two classes of standard JSP test problems: Fisher and Thompson (1963) instances FT06, FT10, FT20, and Lawrence (1984) instances LA01 to LA40.

The proposed algorithm is compared with the following algorithms:

Problem and Heuristic Space
- Storer et al. (1992)

Genetic Algorithms
- Aarts et al. (1994)
- Croce et al. (1995)
- Dorndorf and Pesch (1995)
- Gonçalves and Beirão (1999)

GRASP
- Binato et al. (2002)
- Aiex et al. (2001)

Hybrid Genetic and Simulate Annealing
- Wang and Zheng (2001)

Tabu Search
- Nowicki and Smutnicki (1996)

The experiments were performed using the following configuration:

Population size
The number of chromosomes in the population equals twice the number of operations in the problem
Crossover
The probability of tossing heads is equal to 0.7
Selection
The top 10% from the previous population chromosomes are copied to the next generation
Mutation
The bottom 20% of the population chromosomes are replaced with randomly generated chromosomes
Fitness
Makespan (to minimize)
Seeds
20
Stopping criteria
After 400 generations

The algorithm was implemented in Visual Basic 6.0 and the tests were run on a computer with a 1.333 GHz AMD Thunderbird CPU on the MS Windows Me operating system.

Table 3 summarizes the experimental results. It lists problem name, problem dimension (number of jobs $\times$ number of operations), the best known solution (BKS), the solution obtained by our algorithm (HGA) using parameterized active schedules (HGA-Param. Active), non-delay schedules (HGA-Non-delay) and active schedules (HGA-Active) and the solution obtained by each of the other algorithms. The non-delay and the active schedules were obtained by making the delay of each operation equal to 0 and ∞ respectively.

The total computational time (in seconds) for each problem and for 400 generations of the genetic algorithm and the percentage time spent evolving the population, constructing the schedules and in the local search are presented in Table 4.

Table 3
Experimental results

Inst.	Size	BKS	Param. Active	HGA Non-delay	Active	Wang and Zheng (2001)	Aiex et al. (2003)	Binato et al. (2002)	Now-icki and Smut-nicki (1996)	Gon çal ves and Beirão (1999)	Croce et al. (1995)	Dorndorf & Pesch			Aarts et al.		Storer et al. (1992)
												P-GA (1995)	SBGA (40) (1995)	SBGA (60) (1995)	GLS1 (1994)	GLS2 (1994)	
FT06	6×6	55	55	55	55	55	55	55	55	55		–					
FT10	10×10	930	930	951	945	930	930	938	930	936	946	960			935	945	952
FT20	20×5	1165	1165	1178	1173	1165	1165	1169	1165	1177	1178	1249			1165	1167	
LA01	10×5	666	666	666	666	666	666	666	666	666	666	666	666		666	666	666
LA02	10×5	655	655	665	655		655	655	655	666	666	681	666		668	659	
LA03	10×5	597	597	604	603		597	604	597	597	666	620	604		613	609	
LA04	10×5	590	590	590	598		590	590	590	590	–	620	590		599	594	
LA05	10×5	593	593	593	593		593	593	593	593	–	593	593		593	593	
LA06	15×5	926	926	926	926	926	926	926	926	926	926	926	926		926	926	
LA07	15×5	890	890	890	890		890	890	890	890	–	890	890		890	890	
LA08	15×5	863	863	863	863		863	863	863	863	–	863	863		863	863	
LA09	15×5	951	951	951	951		951	951	951	951	–	951	951		951	951	
LA10	15×5	958	958	958	958		958	958	958	958	–	958	958		958	958	
LA11	20×5	1222	1222	1222	1222	1222	1222	1222	1222	1222	1222	1222	1222		1222	1222	
LA12	20×5	1039	1039	1039	1039		1039	1039	1039	1039	–	1039	1039		1039	1039	
LA13	20×5	1150	1150	1150	1150		1150	1150	1150	1150	–	1150	1150		1150	1150	
LA14	20×5	1292	1292	1292	1292		1292	1292	1292	1292	–	1292	1292		1292	1292	
LA15	20×5	1207	1207	1207	1207		1207	1207	1207	1207	–	1237	1207		1207	1207	
LA16	10×10	945	945	973	947	945	945	946	945	977	979	1008	961	961	9 77	977	981
LA17	10×10	784	784	792	784		784	784	784	787	–	809	787	784	791	791	794
LA18	10×10	848	848	855	848		848	848	848	848	–	916	848	848	856	858	860
LA19	10×10	842	842	851	852		842	842	842	857	–	880	863	848	863	859	860
LA20	10×10	902	907	926	912		902	907	902	910	–	928	911	910	913	916	
LA21	15×10	1046	1046	1079	1074	1058	1057	1091	1047	1047	1097	113 9	1074	1074	1084	1085	
LA22	15×10	927	935	950	962		927	960	927	936	–	998	935	936	954	944	
LA23	15×10	1032	1032	1032	1032		1032	1032	1032	1032	–	1072	1032	1032	1032	1032	
LA24	15×10	935	953	970	955		954	978	939	955	–	1014	960	957	970	981	
LA25	15×10	977	986	1013	1014		984	1028	977	1004	–	1014	1008	1007	1016	1010	
LA26	20×10	1218	1218	1218	1237	1218	1218	1271	1218	1218	1231	1278	1219	1218	1240	1236	
LA27	20×10	1235	1256	1282	1280		1269	1320	1236	1260	–	1378	1272	1269	1308	1300	
LA28	20×10	1216	1232	1250	1250		1225	1293	1216	1241	–	1327	1240	1241	1281	1265	
LA29	20×10	1157	1196	1206	1226		1203	1293	1160	1190	–	1336	1204	1210	1290	1260	
LA30	20×10	1355	1355	1355	1355		1355	1368	1355	1356	–	1411	1355	1355	1402	1386	
LA31	30×10	1784	1784	1784	1784	1784	1784	1784	1784	1784	1784	–			1784	1784	

LA32	30×10	1850	1850	1850	1850		1850	1850	1850	1850	–	–			1850	1850	
LA33	30×10	1719	1719	1719	1719		1719	1719	1719	1719	–	–			1719	1719	
LA34	30×10	1721	1721	1721	1721		1721	1753	1721	1730	–	–			1737	1730	
LA35	30×10	1888	1888	1888	1888		1888	1888	1888	1888	–	–			1894	1890	
LA36	15×15	1268	1279	1303	1313	1292	1287	1334	1268	1305	1305	1373	1317	1317	1324	1311	1305
LA37	15×15	1397	1408	1437	1444		1410	1457	1407	1441	–	1498	1484	1446	1449	1450	1458
LA38	15×15	1196	1219	1252	1228		1218	1267	1196	1248	–	1296	1251	1241	1285	1283	1239
LA39	15×15	1233	1246	1250	1265		1248	1290	1233	1264	–	1351	1282	1277	1279	1279	1258
LA40	15×15	1222	1241	1252	1246		1244	1259	1229	1252	–	1321	1274	1252	1273	1260	1258

Table 4
Computational times

Instance	Size	Total time (s)	% Time		
			Evolutionary process	Schedule construction	Local search
FT06	6×6	13	3.66%	50.65%	45.69%
FT10	10×10	292	3.18%	40.21%	56.60%
FT20	20×5	204	5.07%	54.66%	40.27%
LA01	10×5	37	5.45%	62.01%	32.54%
LA02	10×5	51	5.97%	64.25%	29.78%
LA03	10×5	39	13.26%	48.53%	38.21%
LA04	10×5	42	3.44%	57.13%	39.43%
LA05	10×5	32	13.64%	57.79%	28.58%
LA06	15×5	99	5.96%	61.85%	32.19%
LA07	15×5	86	7.14%	59.61%	33.24%
LA08	15×5	99	4.51%	61.41%	34.08%
LA09	15×5	94	6.36%	65.09%	28.55%
LA10	15×5	91	5.90%	63.67%	30.43%
LA11	20×5	197	4.15%	60.94%	34.91%
LA12	20×5	201	4.03%	59.91%	36.06%
LA13	20×5	189	3.27%	62.39%	34.34%
LA14	20×5	187	4.47%	63.61%	31.92%
LA15	20×5	187	5.30%	58.67%	36.03%
LA16	10×10	232	3.19%	48.21%	48.60%
LA17	10×10	216	4.16%	49.41%	46.43%
LA18	10×10	219	4.19%	48.51%	47.29%
LA19	10×10	235	3.42%	44.53%	52.06%
LA20	10×10	235	4.03%	49.14%	46.83%
LA21	15×10	602	3.17%	45.14%	51.69%
LA22	15×10	629	3.29%	45.53%	51.18%
LA23	15×10	594	3.39%	51.89%	44.72%
LA24	15×10	578	2.99%	46.04%	50.97%
LA25	15×10	609	3.93%	49.85%	46.22%
LA26	20×10	1388	2.52%	44.54%	52.94%
LA27	20×10	1251	2.42%	44.21%	53.37%
LA28	20×10	1267	2.36%	42.77%	54.87%
LA29	20×10	1350	2.38%	42.13%	55.49%
LA30	20×10	1260	2.89%	44.60%	52.52%
LA31	30×10	3745	1.71%	41.92%	56.37%
LA32	30×10	3741	1.72%	44.44%	53.85%
LA33	30×10	3637	1.84%	45.09%	53.07%
LA34	30×10	3615	1.75%	45.40%	52.85%
LA35	30×10	3716	1.76%	42.70%	55.54%
LA36	15×15	1826	2.11%	42.97%	54.92%
LA37	15×15	1860	2.03%	40.60%	57.38%
LA38	15×15	1859	2.17%	39.40%	58.42%
LA39	15×15	1869	1.97%	39.88%	58.15%
LA40	15×15	2185	1.57%	32.98%	65.45%

Table 5 shows the number of instances solved (NIS), and the average relative deviation (ARD), with respect to the BKS. The ARD was calculated for the Hybrid Genetic Algorithm (HGA with parametrized active schedules), and for the other algorithms (OA). The last column (Improvement), presents the reduction in ARD obtained by the genetic algorithm with respect to each of the other algorithms.

Table 5
Average relative deviation to the BKS

Algorithm	NIS	ARD		Improvement
		OA	HGA	HGA
Problem and heuristic space				
Storer et al. (1992)	11	2.44%	0.56%	1.88%
Genetic algorithms				
Aarts et al. (1994)—GLS1	42	1.97%	0.40%	1.57%
Aarts et al. (1994)—GLS2	42	1.71%	0.40%	1.31%
Croce et al. (1995)	12	2.37%	0.07%	2.30%
Dorndorf and Pesch (1995)—PGA	37	4.61%	0.46%	4.15%
Dorndorf and Pesch (1995)—SBGA (40)	35	1.42%	0.48%	0.94%
Dorndorf and Pesch (1995)—SBGA (60)	20	1.94%	0.84%	1.10%
Gonçalves and Beirão (1999)	43	0.90%	0.39%	0.51%
GRASP				
Binato et al. (2002)	43	1.77%	0.39%	1.38%
Aiex et al. (2003)	43	0.43%	0.39%	0.04%
Hybrid genetic and simulated annealing				
Wang and Zheng (2001)	11	0.28%	0.08%	0.20%
Tabu search				
Nowicki and Smutnicki (1996)	43	0.05%	0.39%	−0.34%

Overall, we solved 43 instances with HGA and obtained an ARD of 0.39%. The HGA obtained the best-known solution for 31 instances, i.e. in 72% of problem instances. HGA presented an improvement with respect to almost all others algorithms, the exception being the tabu search algorithm of Nowicki and Smutnicki that had better performance, mainly in the 15×15 problems.

5. Conclusions

This paper presents a hybrid genetic algorithm for the job shop scheduling problem. The chromosome representation of the problem is based on random keys. The schedules are constructed using a priority rule in which the priorities are defined by the genetic algorithm. Schedules are constructed using a procedure that generates parameterized active schedules. After a schedule is obtained, a local search heuristic is applied to improve the solution. The approach is tested on a set of 43 standard instances taken from the literature and compared with 12 other approaches. The computational results show that the algorithm produced optimal or near-optimal solutions on all instances tested. Overall, the algorithm produced solutions with an average relative deviation of 0.39% to the best known solution.

6. Further research

We believe the idea of *parameterized active schedules* could also be applied to the resource constrained project scheduling problem with success.

References

Aarts, E.H.L., Van Laarhoven, P.J.M., Lenstra, J.K., Ulder, N.L.J., 1994. A computational study of local search algorithms for job shop scheduling. ORSA Journal on Computing 6, 118–125.

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

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

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

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

Beasley, D., Bull, D.R., Martin, R.R., 1993. An overview of genetic algorithms: Part 1, Fundamentals, University Computing, Vol. 15, No. 2, pp. 58–69, Department of Computing Mathematics, University of Cardiff, UK.

Binato, S., Hery, W.J., Loewenstern, D.M., Resende, M.G.C., 2002. A GRASP for job shop scheduling. In: Ribeiro, C.C., Hansen, P. (Eds.), Essays and Surveys in Metaheuristics. Kluwer Academic Publishers.

Brucker, P., Jurisch, B., Sievers, B., 1994. A branch and bound algorithm for job-shop scheduling problem. Discrete Applied Mathematics 49, 105–127.

Carlier, J., Pinson, E., 1989. An algorithm for solving the job shop problem. Management Science 35 (29), 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., 1999. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: Hybrid genetic search strategies. Computers & Industrial Engineering 36, 343–364.

Croce, F., Tadei, R., Volta, G., 1995. A genetic algorithm for the job shop problem. Computers and Operations Research 22 (1), 15–24.

Davis, L., 1985. Job shop scheduling with genetic algorithms. In: Proceedings of the First International Conference on Genetic Algorithms and their Applications. Morgan Kaufmann, pp. 136–140.

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

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

French, S., 1982. Sequencing and scheduling—An introduction to the mathematics of the job-shop. Ellis Horwood, John Wiley & Sons, New York.

Giffler, B., Thompson, G.L., 1960. Algorithms for solving production scheduling problems. Operations Research 8 (4), 487–503.

Goldberg, D.E., 1989. Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley.

Gonçalves, J.F., Beirão, N.C., 1999. Um Algoritmo Genético Baseado em Chaves Aleatórias para Sequenciamento de Operações. Revista Associação Portuguesa de Desenvolvimento e Investigação Operacional 19, 123–137 (in Portuguese).

Gonçalves, J.F., Mendes, J.M., 1994. A look-ahead dispatching rule for scheduling operations. VII Latin-Ibero-American Conference on Operations Research and Systems Engineering, University of Chile, Santiago, Chile.

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.

Jain, A.S., Meeran, S., 1999. A state-of-the-art review of job-shop scheduling techniques. European Journal of Operational Research 113, 390–434.

Laarhoven, P.J.M.V., Aarts, E.H.L., Lenstra, J.K., 1992. Job shop scheduling by simulated annealing. Operations Research 40, 113–125.

Lageweg, B.J., Lenstra, J.K., Rinnooy Kan, A.H.G., 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, GSIA, Carnegie Mellon University, Pittsburgh, PA.

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

Lourenço, H.R., 1995. Local optimization and the job-shop scheduling problem. European Journal of Operational Research 83, 347–364.

Lourenço, H.R., Zwijnenburg, M., 1996. Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem. In: Osman, I.H., Kelly, J.P. (Eds.), Metaheuristics: Theory and Applications. Kluwer Academic Publishers, pp. 219–236.

Nowicki, E., Smutnicki, C., 1996. A fast taboo search algorithm for the job-shop problem. Management Science 42 (6), 797–813.

Oliveira, J.A.V., 2000. Aplicação de Modelos e Algoritmos de Investigação Operacional ao Planeamento de Operações em Armaz'ens. Ph.D. Thesis, Universidade do Minho, Portugal (in Portuguese).

Pinson, E., 1995. The job shop scheduling problem: A concise survey and some recent developments. In: Chrëtienne, P., Coffman Jr., E.G., Lenstra, J.K., Liu, Z. (Eds.), Scheduling Theory and its Application. John Wiley and Sons, pp. 277–293.

Roy, B., Sussmann, 1964. Les Problèes d' ordonnancement avec contraintes dijonctives, Note DS 9 bis, SEMA, Montrouge.

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

Spears, W.M., Dejong, K.A., 1991. On the virtues of parameterized uniform crossover. In: Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 230–236.

Storer, R.H., Wu, S.D., Park, I., 1992. Genetic algorithms in problem space for sequencing problems. In: Proceedings of a Joint US–German Conference on Operations Research in Production Planning and Control, pp. 584–597.

Taillard, E.D., 1994. Parallel taboo search techniques for the job shop scheduling problem. ORSA Journal on Computing 6 (2), 108–117.

Vaessens, R.J.M., Aarts, E.H.L., Lenstra, J.K., 1996. Job shop scheduling by local search. INFORMS Journal.

Wang, L., Zheng, D., 2001. An effective hybrid optimisation strategy for job-shop scheduling problems. Computers & Operations Research 28, 585–596.

Williamson, D.P., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevast'janov, S.V., Shmoys, D.B., 1997. Short shop schedules. Operations Research 45 (2), 288–294.