Complex Scheduling Problems Using An Optimization Methodology

JORGE MAGALHÃES-MENDES
Department of Civil Engineering
School of Engineering – Polytechnic of Porto
Rua Dr. António Bernardino de Almeida, 431 – 4200-072 Porto
PORTUGAL
jjm@isep.ipp.pt

Abstract: - Scheduling operations problems arise in diverse areas such as flexible manufacturing, production planning and scheduling, logistics, supply chain problem, etc. A common feature of many of these problems is that no efficient solution algorithms are known that solve each instance to optimality in a time bounded polynomially in the size of the problem, Dorndorf and Pesch [22]. Discrete optimization can help to overcome these difficulties. This paper presents an optimization approach to solve the complex scheduling problem in a job shop environment. This problem is also known as the Job Shop Scheduling Problem (JSSP). The JSSP is a difficult problem in combinatorial optimization for which extensive investigation has been devoted to the development of efficient algorithms. The proposed approach is based on a genetic algorithm technique. Genetic algorithms are an optimization methodology based on a direct analogy to Darwinian natural selection and mutations in biological reproduction. The scheduling rules such as SPT and MWKR are integrated into the process of genetic evolution. The chromosome representation of the problem is based on random keys. The schedules are constructed using a priority rule in which the priorities and delay times of the operations 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 approach.

Key-Words: - Scheduling, manufacturing, heuristics, genetic algorithm, optimization, local search, JSSP.

1 Introduction

A job shop consists of a set of different machines (e.g., lathes, milling machines, drills, etc.) that perform operations on jobs. Each job consists of a sequence of operations, each of which uses one of the machines for a fixed duration. Once started, the operation cannot be interrupted. Each machine can process at most one operation at a time. A schedule is an assignment of operations to time intervals on the machines. The problem is to find a schedule of minimal time to complete all jobs, French [37].

The job shop scheduling problem (JSSP) is one of the most difficult combinatorial optimization problems. The problem finds numerous applications in manufacturing and is central to many supply chain problems that integrate production planning and scheduling. For an extensive treatment of planning and scheduling models and applications in various supply chain settings, see Pinedo [36].

To make the right decision, the manager must have reliable and complete information and adequate scheduling models. Single and small scale productions are characterized by many work orders. So, the application of new tools based on discrete optimization not only in production process but also in determination of priority of technological operations will be appropriate.

There are two categories for job shop scheduling; static and dynamic. In dynamic scheduling, schedules are created during run time and no knowledge of operations is in hand until it arrives. While in static scheduling, schedules are created before run time and can not change. Similarly, operations must all be known in advance. In other words a static job shop scheduling algorithm schedules a set of operations with known processing on machines to optimize some performance metric, such as makespan, communication cost and CPU utilization. In this paper we focus on static scheduling.

The JSSP may be described as follows: n jobs are to be scheduled on m machines. Each job i represents n_i ordered operations. The execution of each operation j of job i (noted as o_{ij}) requires one machine m selected from a set of machines for a fixed duration, see Figure 1. 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.

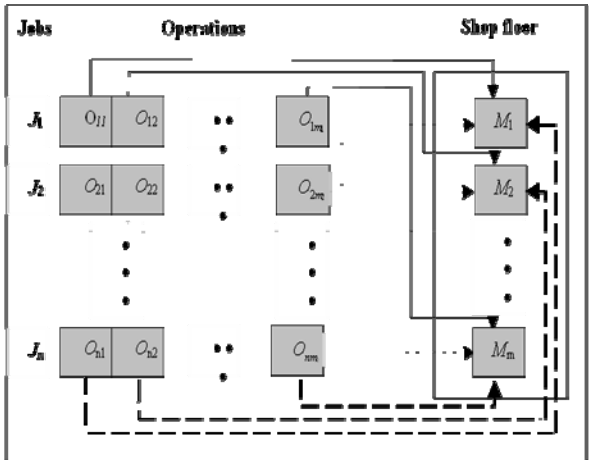

Fig. 1 – A Job shop system.

Let $J = \{0, 1, \ldots, N, N+1\}$ denote the set of all operations to be scheduled and $M = \{1,..., m\}$ 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 $\boldsymbol{F_j}$ represent the finish time of operation j. A schedule, see Fig. 2, can be represented by a vector of finish times ($F_{1, \ldots,} F_m, \ldots, F_{n+1}$). Let $\boldsymbol{A}(t)$ be the set of operations being processed at time t, and let $\boldsymbol{r}_{j,m} = 1$ if operation j requires machine m to be processed and $\boldsymbol{r}_{j,m} = 0$ otherwise.

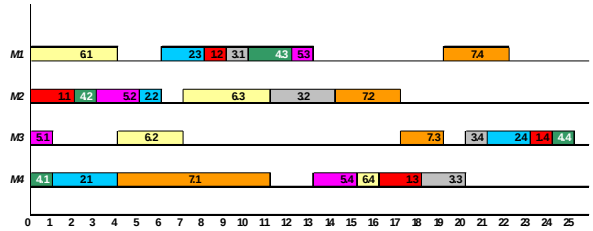

Fig. 2 – A schedule example.

Fig. 2 shows a schedule example with 7 jobs, 4 operations for job and 4 machines.

The conceptual model of the JSSP can be described in the following way:

$$\textbf{Min } \boldsymbol{F}_{n+1} \tag{1}$$

subject to:

$$\boldsymbol{F_k} \le \boldsymbol{F_j} - \boldsymbol{d_j} \qquad \boldsymbol{j} = 1,\ldots,\boldsymbol{N}+1 \;;\; \boldsymbol{k} \in \boldsymbol{P_j} \tag{2}$$

$$\sum_{j \in A(t)} \boldsymbol{r}_{j,m} \le 1 \qquad \boldsymbol{m} \in \boldsymbol{M} \;;\; \boldsymbol{t} \ge 0 \tag{3}$$

$$\boldsymbol{F_j} \ge 0 \qquad \boldsymbol{j} = 1,\ldots,\boldsymbol{N}+1 \tag{4}$$

The objective function (1) minimizes the finish time of operation *N+1* (the last operation), and therefore minimizes the *makespan*. Constraints (2) impose the precedence relations between operations and constraints (3) state that one machine can only process one operation at a time. Finally (4) forces the finish times to be non-negative.

Lenstra and Rinnooy Kan [14] demonstrated that JSSP is NP-hard, so it cannot be exactly solved in a reasonable computation time.

Exact methods (Giffler and Thompson [26], Carlier and Pinson [27, 28], Brucker et al. [29], Williamson et al. [30]) have been successful in solving small instances, including the notorious 10×10 instance of Fisher and Thompson proposed in 1963 and only solved twenty years later. Problems of dimension 15×15 are still considered to be beyond the reach of today's exact methods.

Many approximate methods have been developed in the last two decades to solve the JSSP, such as *simulated annealing* (SA) (Lourenço [16]), *tabu search* (TS) (Nowicki and Smutnicki [12], Pezzela and Merelli [8], Zhang et al. [15]), *genetic algorithms* (GA) (Aarts et al. [19], Croce et al. [21], Dorndorf et al. [22], Gonçalves and Beirão [23], Wang and Zheng [24], Zhou et al. [32], Essafi et al. [4], Gonçalves et al. [5], Choi and Park [33], Chiu et al. [34, 35]), *particle swarm optimization* (PSO) (Sha and Hsu [7]), *greedy randomized adaptive search procedure* (GRASP), Binato et al. [25], Aiex et al. [20]) and Rego and Duarte [9] proposed a filter-and-fan approach based on the *shifting bottleneck procedure* (SBP).

In this paper, we present a new optimization approach for the job shop scheduling problem. The remainder of the paper is organized as follows: in Section 2 the types of schedules, Section 3 the approach, Section 4 the genetic algorithm, Section 5 the schedule generation and Section 6 the local

search. In Section 7 we test the optimization approach on Fisher and Thompson [17] and Lawrence [18] test problems. Finally, conclusion and remarks for further works are given in Section 8.

2 Types of schedules

Classifying schedules is the basic work to be done before attacking scheduling problems [39].

Schedules can be classified into one of the following three types of schedules:

i) *Feasible* schedules. A schedule is said to be feasible if it is non-preemptive and if the precedence and resource constraints are satisfied.
ii) *Semi-active schedules*. These are feasible schedules obtained by scheduling operations as early as possible. In a semi-active schedule the start time of a particular operation is constrained by the processing of a different operation on the same resource or by the processing of the directly preceding operation on a different resource.
iii) *Active schedules*. These are feasible schedules in which no operation could be started earlier without delaying some other operation or breaking a precedence constraint. Active schedules are also semi-active schedules. An optimal schedule is always active.
iv) *Non-delay schedules*. These are feasible schedules in which no resource is kept idle at a time when it could begin processing some operation. Non-delay schedules are active and hence are also semi-active.

The set of active schedules is usually very large and contains many schedules with poor quality. To reduce the solution space we use the concept of parameterized active schedules.

The concept of parameterized active schedules is proposed in Gonçalves and Beirão [23], Mendes [10], Gonçalves et al. [5] and Mendes et al. [6]. This type of schedule consists of schedules in which no resource is kept idle for more than a predefined period if it could start processing some operation. If the predefined period is set to zero, then we obtain a non-delay schedule.

3 The Approach

The approach combines a genetic algorithm, a schedule generation scheme 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.

For each chromosome the following three phases are applied:

1. Schedule parameters - this phase is responsible for transforming the chromosome supplied by the genetic algorithm into the priorities of the operations and delay time;
2. Schedule generation - this phase makes use of the priorities and the delay time and constructs active schedules;
3. 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 quality is feedback to the genetic algorithm. Figure 3 illustrates the sequence of steps applied to each chromosome. Details about each of these phases will be presented in the next sections.

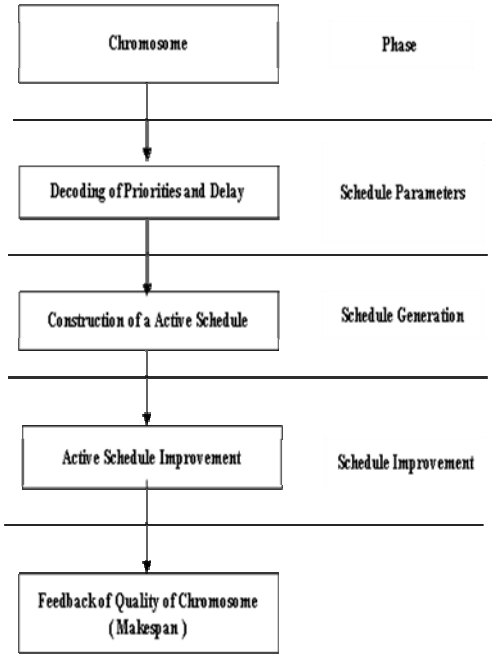

Fig. 3 - Phases of the proposed approach.

4 Application of genetic algorithm

The approach presented in this paper is based on a genetic algorithm to perform its optimization process.

Genetic algorithms (GAs) are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search

algorithm with some of the innovative flair of human search [1, 2].

One fundamental advantaged of GAs from traditional methods is described by Goldberg [1]: in many optimization methods, we move gingerly from a single solution in the decision space to the next using some transition rule to determine the next solution. This solution-to-solution method is dangerous because it is a perfect prescription for locating false peaks in multimodal search spaces. By contrast, GAs work from a rich database of solutions simultaneously (a population of chromosomes), climbing many peaks in parallel; thus the probability of finding a false peak is reduced over methods that go solution to solution.

The general schema of GA may be illustrated as follows (Fig. 4).

```
procedure GENETIC-ALGORITHM

Generate initial population P0;
 Evaluate population P0;
 Initialize generation counter g ←0;

 While stopping criteria not satisfied repeat
    Select some elements from Pg to copy into Pg+1;
    Crossover some elements of Pg and put into Pg+1;
    Mutate some elements of Pg and put into Pg+1;
    Evaluate some elements of Pg and put into Pg+1;
    Increment generation counter: g ← g+1;
  End while

End GENETIC-ALGORITHM;
```

Fig.4 - Pseudo-code of a genetic algorithm.

First of all, an initial population of potential solutions (individuals) is generated randomly. A selection procedure based on a fitness function enables to choose the individuals candidate for reproduction. The reproduction consists in recombining two individuals by the crossover operator, possibly followed by a mutation of the offspring. Therefore, from the initial population a new generation is obtained. From this new generation, a second new generation is produced by the same process and so on. The stop criterion is normally based on the number of generations.

4.1 Decoding

The genetic algorithm uses a random key alphabet which is comprised of real random numbers between 0 and 1.

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 $2n$ genes where n is the number of operations (excluding 0 and n+1):

$$Chromosome = (gene_1, .., gene_n, gene_{n+1}, ..., gene_{2n})$$

4.1.1 Decoding the priorities of operations

The priority decoding expression used the following expression

$$\boldsymbol{PRIORITY_j = gene_j} \qquad \boldsymbol{j} = 1,...,\boldsymbol{n}.$$

4.1.2 Decoding the delay times

The genes between n+1 and $2n$ are used to determine the delay times used when scheduling an operation. The delay time used by each scheduling iteration g, Delay_g, is given by the following expression:

$$\text{Delay}_g = \text{gene}_{n+g} \times 1.5 \times \text{MaxDur}$$

where MaxDur is the maximum duration of all operations. The factor 1.5 was obtained after some experimental tuning.

4.2 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.

Reproduction is a process in which individual (chromosome) is copied according to their fitness values (*makespan*).

Reproduction is accomplished by first copying some of the best individuals from one generation to the next, in what is called an elitist strategy.

In this paper the fitness proportionate selection, also known as roulette-wheel selection, is the genetic operator for selecting potentially useful solutions for reproduction. The characteristic of the roulette wheel selection is stochastic sampling.

The fitness value is used to associate a probability of selection with each individual chromosome. If f_i is the fitness of individual i in the population, its probability of being selected is,

$$\boldsymbol{p_i} = \frac{\boldsymbol{f_i}}{\sum_{\boldsymbol{i}=1}^{\boldsymbol{N}} \boldsymbol{f_i}} \quad , \qquad \boldsymbol{i} = 1,...,\boldsymbol{n} \qquad (5)$$

An example is presented in Table 1.

A roulette wheel model is established to represent the survival probabilities for all the individuals in the

population. Then the roulette wheel is rotated for several times [1], see Fig. 5.

After selection the mating population consists of the chromosomes (individuals): 1, 2, 3, 4, 5 and 6.

Number of chromosome	Fitness value	Selection probability
1	14	0,20
2	12	0,17
3	10	0,14
4	9	0,13
5	8	0,11
6	7	0,10
7	4	0,06
8	3	0,04
9	2	0,03
10	1	0,01

Table 1: Selection probability and fitness value.

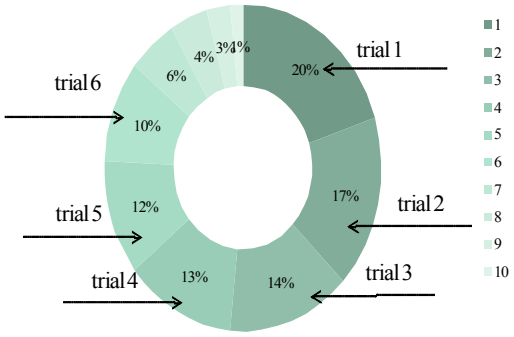

Fig. 5 - Roulette-wheel selection.

After selecting, crossover may proceed in two steps. First, members of the newly selected (reproduced) chromosomes in the mating pool are mated at random. Second, each pair of chromosomes undergoes crossover as follows: an integer position *k* along the chromosome is selected uniformly at random between 1 and the chromosome length *l*. Two new chromosomes are created swapping all the genes between *k*+1 and *l* [1], see Fig. 6.

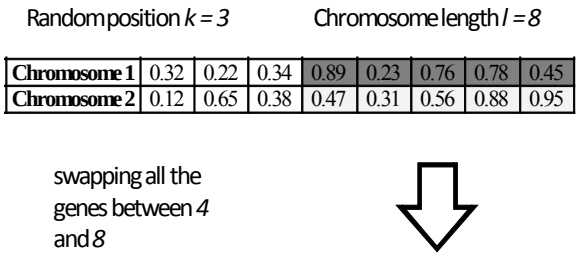

Fig.6 – Crossover operator example.

The mutation operator preserves diversification in the search. This operator is applied to each offspring in the population with a predetermined probability. We assume that the probability of the mutation in this paper is 0.001. With 60 genes positions we should expect 60 x 0.001 = 0.06 genes to undergo mutation for this probability value.

This evolutionary strategy was applied to the resource constrained project scheduling problem with a good performance, see Mendes [3, 31].

Fig. 7 shows this evolutionary strategy with the operators selection, recombination (or crossover) and mutation.

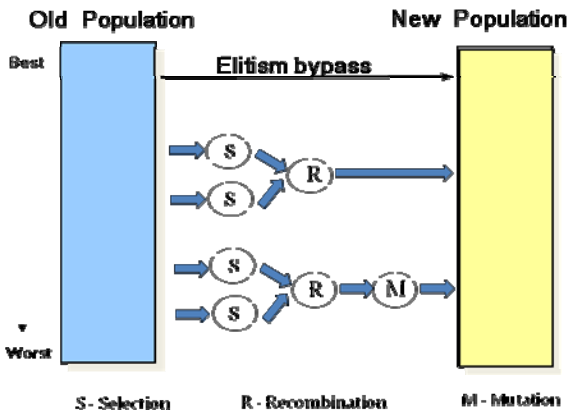

Fig.7 – Evolutionary strategy.

4.3 Initial population

Through the initialization process the initial population will be generated usually randomly from the range of possible solutions (chromosomes). The quality of this population is initially poor and one way to improve is to incorporate some chromosomes generated by selected priority rules.

In this work are selected the priority rules MWKR (the job with the most work remaining would be scheduled first) and SPT (the job with the shortest

processing time has the highest priority for processing) to improve some chromosomes of the initial population.

5 Schedule Generation

Schedule generation schemes (SGS) are the core of most heuristic solution procedures for the JSSP. SGS start from scratch and build a feasible schedule by stepwise extension of a partial schedule. A partial schedule is a schedule where only a subset of the n+2 operations have been scheduled. There are two different classics methods SGS available. They can be distinguished into operation and time incrementation. The so called serial SGS performs operation-incrementation and the so called parallel SGS performs time-incrementation [38].

The constructive heuristic used to construct active schedules is based on a scheduling generation scheme that does time incrementing, called *parallel modified*.

This heuristic makes use of the priorities and the delay times defined by the genetic algorithm and constructs active schedules. This heuristic is described by Mendes [3], Gonçalves et al. [5] and Mendes et al. [6].

Fig. 8 illustrates where the set of parameterized active schedules is located relative to the class of semi-active, active, and non-delay schedules.

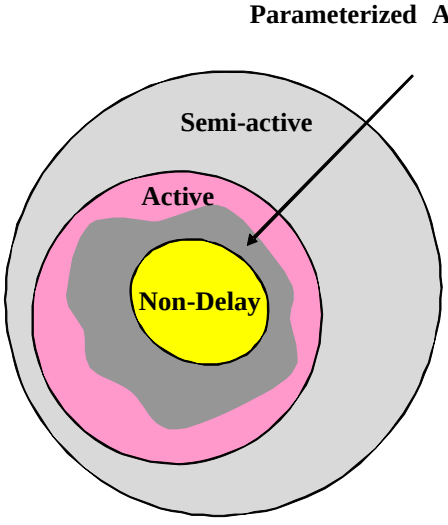

Fig. 8 – Parameterized active schedules.

The heuristic used to construct parameterized active schedules is based on a scheduling generation scheme that does time incrementing. For each iteration $\boldsymbol{g}$, there is a scheduling time $\boldsymbol{t_g}$. The active set comprises all operations which are active at $\boldsymbol{t_g}$, i.e.

$$\boldsymbol{A_g} = \left\{ \boldsymbol{j} \in \boldsymbol{J} \,|\, \boldsymbol{F_j} - \boldsymbol{d_j} \le \boldsymbol{t_g} < \boldsymbol{F_j} \right\}.$$

Initialization: $\boldsymbol{g}=1,\ \boldsymbol{t_1}=0,\ \boldsymbol{A_0}=\{0\},\ \Gamma_o=\{0\},\ \boldsymbol{S_0}=\{0\},\ \boldsymbol{RD_k}(0)=\boldsymbol{R_k}\ \ (\boldsymbol{k}\in\boldsymbol{K})$

while $|\boldsymbol{S_g}| < \boldsymbol{n}+2$ **repeat**

{

Update $\boldsymbol{E_g}$

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

{

Select operation with highest priority

$$\boldsymbol{j^*} = \underset{\boldsymbol{j}\in\boldsymbol{E_g}}{\operatorname{argmax}} \left\{ \boldsymbol{PRIORITY_j} \right\}$$

Calculate earliest finish time (in terms of precedence only)

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

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

$$\boldsymbol{F_{j^*}} = \min \left\{ \boldsymbol{t} \in \left[\boldsymbol{FMC_{j^*}} - \boldsymbol{d_{j^*}}, \infty \right[\cap \Gamma_g \ \middle|\ \boldsymbol{r_{j^*,k}} \le \boldsymbol{RD_k}(\tau), \ \boldsymbol{k}\in\boldsymbol{K} \,|\, \boldsymbol{r_{j^*,k}} > 0,\ \tau \in \left[\boldsymbol{t},\ \boldsymbol{t} + \boldsymbol{d_{j^*}} \right] \right\} + \boldsymbol{d_{j^*}}$$

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

Iteration increment: $\boldsymbol{g = g+1}$

Update $\boldsymbol{A_g}, \boldsymbol{E_g}, \boldsymbol{RD_k(t)} \,|\, \boldsymbol{t} \in \left[\boldsymbol{F_{j^*}} - \boldsymbol{d_{j^*}}, \boldsymbol{F_{j^*}} \right], \boldsymbol{k}\in\boldsymbol{K} \,|\, \boldsymbol{r_{j^*,k}} > 0$

}

Determine the time associated with operation $\boldsymbol{g}$

$$\boldsymbol{t_g} = \min \left\{ \boldsymbol{t} \in \Gamma_{g-1} \,|\, \boldsymbol{t} > \boldsymbol{t_{g-1}} \right\}$$

}

Fig. 9 - Pseudo-code to construct parameterized active schedules, Mendes [10]

The remaining resource capacity of resource k at instant time $\boldsymbol{t_g}$ is given by

$$\boldsymbol{RD_k}\left(\boldsymbol{t_g}\right) = \boldsymbol{R_k}\left(\boldsymbol{t_g}\right) - \sum_{\boldsymbol{j}\in\boldsymbol{A_g}} \boldsymbol{r_{j,k}}$$

Sg comprises all operations which have been scheduled up to iteration $\boldsymbol{g}$, and $\boldsymbol{F_g}$ comprises the finish times of the operations in $\boldsymbol{S_g}$. Let $\boldsymbol{Delay_g}$ be the delay time associated with iteration $\boldsymbol{g}$, and let $\boldsymbol{E_g}$ comprise all operations which are precedence feasible in the interval $[\boldsymbol{t_g}, \boldsymbol{t_g} + \boldsymbol{Delay_g}]$, i.e.

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

The set $\boldsymbol{E_g}$ is responsible for forcing the selection to be made only amongst operations which will have a delay smaller or equal to the maximum allowed delay.

The parameters $PRIORITY_j$ and $Delay_g$ (priority of operation j and delays) are supplied by the genetic algorithm.

The algorithmic description of the scheduling generation scheme used to create parameterized active schedules is given by the pseudo-code shown in Figure 9.

6 Local Search

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 improve the solution quality.

In this work is applied the two exchange local search, based on the disjunctive graph model of Roy and Sussmann [11] and the neighborhood of Nowicki and Smutnicki [12], see Gonçalves et al. [5] and Mendes [10].

7 Computational results

The experiments were performed on an Intel Core 2 Duo CPU T7250 @2.00 GHz. The algorithm was coded in Visual Basic 6.0. The performance of the GA-RKV-JSP (Genetic Algorithm - Random Key Variant) was evaluated on a standard set of 43 benchmark instances belonging to two classical sets known as FT06, FT10, FT20 from Fisher and Thompson [17] and LA01-LA40 from Lawrence [18]. The problem size varies between 6 and 30 jobs and between 5 and 15 machines.

7.1 Genetic algorithm configuration

Though there is no straightforward way to configure the parameters of a genetic algorithm, we obtained good results with values: **population size** of 2 × number of operations in the problem; **mutation probability** of 0.001; **top** (best) 1% from the previous population chromosomes are copied to the next generation; **stopping criterion** of 400 generations; **initial population**: 1% chromosomes calculated by priority rules MWKR and SPT.

7.2 Experimental results

Table 2 summarizes the experimental results. It lists problem name, problem dimension (number of jobs x number of operations), best known solution and the solution obtained by the approach proposed $C_{\max_i}$ (*makespan*).

Instance	*Size*	*BKS*	*Makespan*
FT06	6x6	55	55
FT10	10x10	930	930
FT20	20x5	1165	1165
LA01	10x5	666	666
LA02	10x5	655	655
LA03	10x5	597	597
LA04	10x5	590	590
LA05	10x5	593	593
LA06	15x5	926	926
LA07	15x5	890	890
LA08	15x5	863	863
LA09	15x5	951	951
LA10	15x5	958	958
LA11	20x5	1222	1222
LA12	20x5	1039	1039
LA13	20x5	1150	1150
LA14	20x5	1292	1292
LA15	20x5	1207	1207
LA16	10x10	945	945
LA17	10x10	784	784
LA18	10x10	848	848
LA19	10x10	842	842
LA20	10x10	902	907
LA21	15x10	1046	1056
LA22	15x10	927	937
LA23	15x10	1032	1032
LA24	15x10	935	949
LA25	15x10	977	984
LA26	20x10	1218	1218
LA27	20x10	1235	1256
LA28	20x10	1216	1231
LA29	20x10	1157	1194
LA30	20x10	1355	1355
LA31	30x10	1784	1784
LA32	30x10	1850	1850
LA33	30x10	1719	1719
LA34	30x10	1721	1721
LA35	30x10	1888	1888
LA36	15x15	1268	1278
LA37	15x15	1397	1408
LA38	15x15	1196	1213
LA39	15x15	1233	1248
LA40	15x15	1222	1237

Table 2: Experimental results.

Table 3 shows the number of instances solved (NIS) and the average relative deviation (ARD) with respect to the best known solution (BKS).

For comparison to other methods, we have used one measure, namely the *average relative deviation* (ARD):

$$RE = \sum_{i=1}^{NIS} \frac{C_{\max_i} - BKS_i}{BKS_i} \quad (6)$$

$$ARD = \frac{RE}{NIS} \quad (7)$$

Table 3, column 4, shows the ranking of the best ten approaches that solved the test problems used in this computational study.

The computational time dispended is in the range [7, 3200] seconds.

Algorithm	*NIS*	*ARD*	*RANK*
Genetic Algorithms			
Aarts et al. [19] - GLS1	42	1.97%	-
Aarts et al. [19] - GLS2	42	1.71%	-
Croce et al. [21]	12	2.37%	-
Dorndorf et al. [22] - PGA	37	4.61%	-
Dorndorf et al. [22] - SBGA (40)	35	1.42%	-
Dorndorf et al. [22] - SBGA (60)	20	1.94%	-
Gonçalves and Beirão [23]	43	0.90%	10
Gonçalves et al. [5]	43	0.39%	8
This paper	**43**	**0.38%**	**7**
GRASP			
Binato et al. [25]	43	1.77%	-
Aiex et al. [20]	43	0.43%	9
Tabu Search			
Nowicki and Smutnicki [12]	43	0.05%	3
F. Pezella and E. Merelli [8]	43	0.10%	4
Zhang et al. [15]	43	0.00%	1
PSO			
Sha and Hsu [7] - HPSO	43	0.02%	2
Sha and Hsu [7] - PSO	43	0.37%	6
SBP & LNS			
Rego and Duarte [9]	43	0.29%	5

Table 3: Top-ten computational results for FT and LA test problems.

8 Conclusion and further work

This paper presents a discrete optimization approach for the JSSP. This approach is based on a genetic algorithm. The chromosome representation of the problem is based on random keys. Reproduction, crossover and mutation are applied to successive chromosome populations to create new chromosome populations. These operators are simplicity itself, involving random number generation, chromosome copying and partial chromosome exchanging. The scheduling rules such as SPT and MWKR are integrated into the process of genetic evolution.

The schedules are constructed using a priority rule in which the priorities for each operation are defined by the genetic algorithm. Schedules are constructed using a constructive heuristic. After a schedule is obtained, a local search heuristic is applied to improve the solution.

The approach was tested on a set of 43 standard instances taken from the literature and compared with the best state-of-the-art approaches. The algorithm produced good results when compared with other approaches.

Further work could be conducted to explore the possibility of genetically correct the chromosomes supplied by the genetic algorithm to reflect the solutions obtained by the local search heuristic.

Acknowledgements

This work has been partially supported by the FCT – Portuguese Foundation for the Science and Technology.

References:

[1] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, 1989.

[2] D. Beasley, D.R. Bull and R.R. Martin, *An Overview of Genetic Algorithms: Part 1, Fundamentals*, University Computing, Department of Computing Mathematics, University of Cardiff, UK, Vol. 15(2), 1993, pp. 58-69.

[3] J. Magalhães-Mendes, Project scheduling under multiple resources constraints using a genetic algorithm, *WSEAS TRANSACTIONS on BUSINESS and ECONOMICS*, Issue 11, Volume 5, November 2008, pp. 487-496.

[4] I. Essafi, Y. Mati and S.D. Pérès, A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem, *Computers & Operations Research*, Vol. 35, Issue 8, 2008, pp. 2599-2616.

[5] J.F. Gonçalves, J.M. Mendes, and M.C.G. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, Vol. 167, 2005, pp. 77-95.

[6] J.J.M. Mendes, J.F. Gonçalves and M.G.C. Resende, A random key based genetic algorithm for the resource constrained project scheduling

problem, *Computers & Operations Research*, Vol. 36, 2009, pp. 92-109.

[7] D. Y. Sha and C. Hsu, A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering,* 51, 4, 2006, pp. 791-808.

[8] F. Pezzela and E. Merelli, A tabu search method guided by shifting bottleneck for the job shop scheduling problem, *European Journal of Operational Research*, Vol. 120, 2000, pp. 297-310.

[9] C. Rego and R. Duarte, A filter-and-fan approach to the job shop scheduling problem, *European Journal of Operational Research*, Vol. 194, 2009, pp. 650–662.

[10] J.J.M. Mendes, *Sistema de Apoio à Decisão para Planeamento de Sistemas de Produção do Tipo Projecto*, Ph.D. Thesis, Departamento de Engenharia Mecânica e Gestão Industrial, Faculdade de Engenharia da Universidade do Porto, Portugal, 2003. (*In portuguese*)

[11] B.Roy and Sussmann, Les Problèmes d'ordonnancement avec contraintes dijonctives, *Note DS 9 bis, SEMA*, Montrouge, 1964.

[12] E. Nowicki and C. Smutnicki, A Fast Taboo Search Algorithm for the Job-Shop Problem, *Management Science*, Vol. 42, No. 6, 1996, pp. 797-813.

[13] S. Binato, W.J. Hery, D.M. Loewenstern and M.G.C. Resende, *A GRASP for Job Shop Scheduling. In: Essays and Surveys in Metaheuristics*, Ribeiro, Celso C., Hansen, Pierre (Eds.), Kluwer Academic Publishers, 2002.

[14] J.K. Lenstra and A.H.G. Rinnoy Kan, Computational complexity of discrete optimisation problems, *Annals of Discrete Mathematics*, Vol. 4, 2002, pp. 121-140.

[15] C. Y. Zhang, P. Li and Z. Guan, A very fast TS/SA algorithm for the job shop scheduling problem, *Computers & Operations Research*, Vol. 35, 2008, pp. 282-294.

[16] H.R. Lourenço, Local optimization and the job-shop scheduling problem, *European Journal of Operational Research*, Vol. 83, 1995, pp. 347-364.

[17] H. Fisher and G.L. Thompson, *Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules*, in: Industrial Scheduling, J.F. Muth and G.L. Thompson (eds.), Prentice-Hall, Englewood Cliffs, NJ, 1963, pp. 225-251.

[18] S. Lawrence, *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1984.

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

[20] R.M.Aiex, S.Binato and M.G.C. Resende, Parallel GRASP with Path-Relinking for Job Shop Scheduling, *Parallel Computing*, Vol. 29, Issue 4, 2003, pp. 393 - 430.

[21] F. Croce, R. Tadei, and G. Volta, A Genetic Algorithm for the Job Shop Problem, *Computers and Operations Research*, Vol. 22(1), 1995, pp. 15-24.

[22] U. Dorndorf, and E. Pesch, Evolution Based Learning in a Job Shop Environment, *Computers and Operations Research*, Vol. 22, 1995, pp. 25-40.

[23] J.F. Gonçalves and N.C. Beirão, Um Algoritmo Genético Baseado em Chaves Aleatórias para Sequenciamento de Operações, *Revista Associação Portuguesa de Desenvolvimento e Investigação Operacional*, Vol. 19, 1999, pp. 123-137, (in Portuguese).

[24] L. Wang, and D. Zheng, An effective hybrid optimisation strategy for job-shop scheduling problems, *Computers & Operations Research*, Vol. 28, 2001, pp. 585-596.

[25] S. Binato, W.J.Hery, D.M. Loewenstern and M.G.C.Resende, *A GRASP for Job Shop Scheduling*. In: Essays and Surveys in Metaheuristics, Ribeiro, Celso C., Hansen, Pierre (Eds.), Kluwer Academic Publishers, 2002.

[26] B. Giffler and G.L. Thompson, Algorithms for Solving Production Scheduling Problems, *Operations Research*, Vol. 8(4), 1960, pp. 487-503.

[27] J. Carlier and E. Pinson, An Algorithm for Solving the Job Shop Problem. *Management Science*, Feb, 35(29), 1989, pp.164-176.

[28] J. Carlier and E. Pinson, A practical use of Jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, Vol. 26, 1990, pp. 269-287.

[29] P. Brucker, B. Jurisch and B. Sievers, A Branch and Bound Algorithm for Job-Shop Scheduling Problem, *Discrete Applied Mathematics*, Vol. 49, 1994, pp. 105-127.

[30] D. P.Williamson, L.A. Hall, J.A. Hoogeveen, , C. A. J.Hurkens, J. K. Lenstra, S. V. Sevastjanov and D. B. Shmoys, Short Shop Schedules, *Operations Research*, 45(2), 1997, pp. 288-294.

[31] J. Magalhães-Mendes, Project scheduling using a competitive genetic algorithm. In *Proceedings of the 8th Conference on Simulation, Modelling and Optimization* (Santander, Cantabria, Spain, September 23 - 25, 2008). J. M. de la Maza and P.

L. Espí, Eds. Mathematics And Computers In Science And Engineering. WSEAS, Stevens Point, Wisconsin, 2008, pp. 39-42.

[32]H. Zhou, Y. Feng and L. Han, The hybrid heuristic genetic algorithm for job shop scheduling, *Computers & Industrial Engineering*, Vol. 40, 2001, pp. 191-200.

[33]H.R. Choi and B.J. Park, Genetic Algorithm for the Integration of Process Planning and Scheduling in a Job Shop, *WSEAS TRANSACTIONS on INFORMATION SCIENCE & APPLICATIONS*, Issue 12, Volume 3, December 2006, pp. 2498-2504.

[34]H. Chiu, K. Hsieh, Y.T. Tang and W. Chien, Employing a Genetic Algorithm Based on Knowledge to Address the Job Shop Scheduling Problem, *WSEAS TRANSACTIONS on COMPUTER RESEARCH*, Issue 2, Volume 2, February 2007, pp. 327-333.

[35]H. Chiu, K. Hsieh, Y.T. Tang and C.Y. Wang, A Novel Approach to Address the Job-Shop Scheduling Problem by using a Tabu Genetic Algorithm, *WSEAS TRANSACTIONS on COMPUTER RESEARCH*, Issue 2, Volume 2, February 2007, pp. 339-345.

[36]M.L. Pinedo, Planning and Scheduling in Manufacturing and Services, Series in Operations Research and Financial Engineering, Springer-Verlag, 2006.

[37] S. French, Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop, Horwood, Chichester, UK, 1982.

[38]R.Kolisch and S.Hartmann, Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis, J. Weglarz (editor), Kluwer, Amsterdam, the Netherlands, 1999, pp. 147–178.

[39]R. Kolisch, *Project Scheduling under Resource Constraints*, Physica-Verlag, Germany, 1995.