A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem*

José Fernando Gonçalves
Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Mecânica e Gestão Industrial
Rua Dr. Roberto Frias
4200-465 Porto, Portugal
jfgoncal@fe.up.pt

Jorge José de Magalhães Mendes
Instituto Superior de Engenharia - Instituto Politécnico do Porto
Departamento de Engenharia Informática
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto, Portugal
jmendes@fe.up.pt

Maurício G.C. Resende
Internet and Network Systems Research
AT&T Labs Research, Florham
Park, NJ 07932 USA
mgcr@research.att.com

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.

* AT&T Labs Research Technical Report TD-5EAL6J, September 2002.

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.

Let $J = \{0, 1, \ldots, n, n+1\}$ denote the set of 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 can be represented by a vector of finish times ($F_{1,}$, F_m, ... , 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.

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

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

Subject to:

$$\boldsymbol{F}_k \leq \boldsymbol{F}_j - \boldsymbol{d}_j \qquad \boldsymbol{j} = 1, \ldots, \boldsymbol{n}+1 \quad ; \quad \boldsymbol{k} \in \boldsymbol{P}_j \tag{2}$$

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

$$\boldsymbol{F}_j \geq 0 \qquad \boldsymbol{j} = 1, \ldots, \boldsymbol{n}+1 . \tag{4}$$

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.

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.

* AT&T Labs Research Technical Report TD-5EAL6J, September 2002.

Exact methods (Giffler and Thompson (1960), Carlier and Pinson (1989, 1990), Applegate and Cook (1991), Brucker et al. (1994), Williamson et al. (1997)) 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. For such problems there is a need for good heuristics. Surveys of heuristic methods for the JSP are given in Pinson (1995), Vaessens et al. (1996) and Cheng et al. (1999). These include dispatching rules reviewed in French (1982), Gray and Hoesada (1991), Gonçalves and Mendes (1994), the shifting bottleneck approach (Adams et al. (1988) and Applegate and Cook (1991)), local search (Vaessens et al. (1996), Lourenço (1995) and Lourenço and Zwijnenburg (1996)), simulated annealing (Lourenço (1995), Laarhoven et al. (1992)), tabu search (Taillard (1994), Lourenço and Zwijnenburg (1996), and Nowicki and Smutnicki (1996)), and genetic algorithms (Davis (1985), Storer et al. (1992), Aarts et al. (1994), Croce et al. (1995), Dorndorf et al. (1995), Gonçalves and Beirão (1999), and Oliveira (2000)). Recently, Binato et al. (2002) described a greedy randomized adaptive search procedure (GRASP), Aiex et al. (2001) described a parallel GRASP with path-relinking, and Wang and Zheng (2001) described a hybrid optimization strategy for JSP. A comprehensive survey of job shop scheduling techniques can be found in Jain and Meeran (1999).

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. Types of Schedules

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

- **Semi-active schedule**: These feasible schedules are obtained by sequencing operations as early as possible. In a semi-active schedule, no operation can be started earlier without altering the processing sequences.

- **Active schedule:** These feasible schedules are 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, so the search space can be safely limited to the set of all active schedules.

- **Non-delay schedule**: These feasible schedules are schedules in which no machine is kept idle when it could start processing some operation. Non-delay schedules are necessarily active and hence also necessarily semi-active.

Later in this paper we will use parameterized active schedules (Gonçalves and Beirão (1999)). This type of schedule consists of schedules in which no machine is kept idle for more than a predefined value if it could start processing some operation. If the predefined value is set to zero, then we obtain a non-delay schedule. The basis concept of this type of schedule is presented in the next section.

2.1 Parameterized Active Schedules

As mentioned above, 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 poor quality in terms of makespan. In order to reduce the solution space and to control the delay times, we used the concept of parameterized active schedules (Gonçalves and Beirão (1999)).

Figure 1 illustrates where the set of parameterized active schedules is located relative to the class of semi-active, active, and non-delay schedules. By controlling the maximum delay time allowed, one can reduced or increased this solution space. A maximum delay time equal to zero is equivalent to restricting the solution space to non-delay schedules.

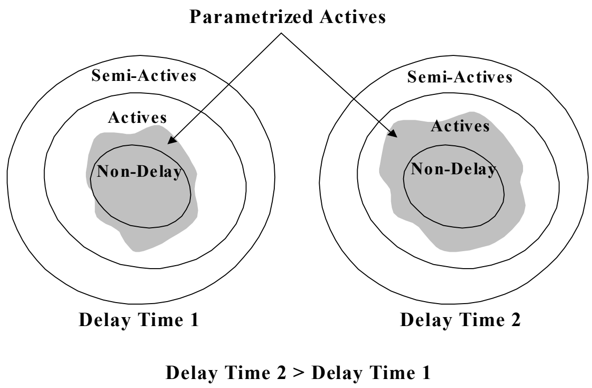

Figure 1 – Parameterized active schedules.

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

3. New Approach for Job Shop Scheduling

The new approach combines a genetic algorithm, a schedule generator procedure that generates parameterized active schedules, and a local search procedure. The approach consists in the following three phases:

- *Assignment of priorities and delay times to the operations*. This phase makes use of a genetic algorithm to define and evolve the priorities of the operations and delay times.

- *Construction procedure*. This phase makes use of the priorities and the delay times defined in the first phase, and constructs parameterized active schedules.

- *Local search procedure*. This phase is used to improve the solution obtained by the construction procedure.

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 Figure 2).

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 terminology, 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.

Genetic Algorithm
{
    Generate initial population $\boldsymbol{P}_t$
    Evaluate population $\boldsymbol{P}_t$
    While stopping criteria not satisfied **Repeat**
    {
        Select elements from $\boldsymbol{P}_t$ to copy into $\boldsymbol{P}_{t+1}$
        Crossover elements of $\boldsymbol{P}_t$ and put into $\boldsymbol{P}_{t+1}$
        Mutation elements of $\boldsymbol{P}_t$ and put into $\boldsymbol{P}_{t+1}$
        Evaluate new population $\boldsymbol{P}_{t+1}$
        $\mathbf{P}_t = \boldsymbol{P}_{t+1}$
    }
}

Figure 2 - A standard genetic algorithm.

3.1.1 Chromosome Representation and Decoding

The genetic algorithm described in this paper uses a random key alphabet **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 $2n$ genes where n is the number of operations.

$$Chromosome = (gene_1, gene_2, ..., gene_n, \; gene_{n+1}, ..., gene_{2n})$$

The first n genes are used as operations priorities, i.e.

$$Priority_j = Gene_j.$$

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 calculated by the following expression:

$$Delay_g = gene_g \times 1.5 \times MaxDur,$$

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

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 mimimum. 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. Figure 3 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). Figure 3 shows one potential crossover outcome:

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*

Figure 3 - Example of Parameterized Uniform crossover.

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.

Figure 4 depicts the transitional process between two consecutive generations.

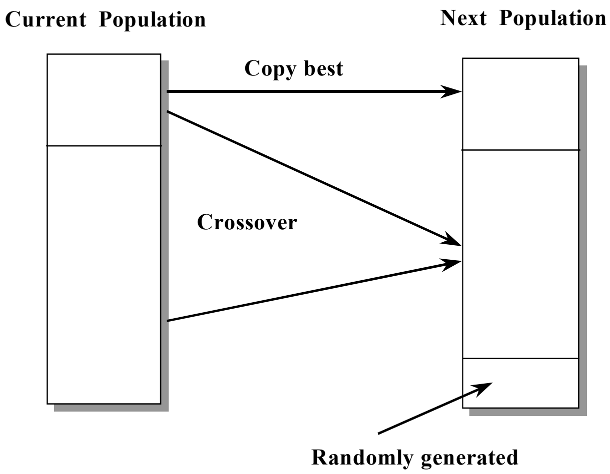

Figure 4- Transitional process between consecutive generations.

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. $A_g(t_g)=\{ j\in J \mid F_j - d_j \le t_g < F_j \}$. 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.

$$E_g(t_g, Delay_g)=\{ j\in J \setminus S_g \mid F_i \le t_g + Delay_g \ \ (i\in P_j) \}.$$

The algorithmic description of the scheduling generation scheme used to generate parameterized active schedules is given by pseudo-code shown in Figure 5.

Initialization: g=0, t_g=0, A_0={0}, $RMC_m(0)$= 1, $F_g(0)$ ={0}, $S_g(0)$ ={0}

while $|S_g| \le n+1$ **repeat**

{

Iteration increment

$g = g+1$

Determine the time associated with iteration g

$t_g = Min_{j\in A_g}\{F_j\}$

Calculate $A_g(t_g)$, $RMC_m(t_g)$, $E_g = E_g(t_g, Delay_g)$

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

{

Select operation with highest priority

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

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)

$$F_{j^*} = \min\Big\{ t\in\big[EF_{j^*} - d_{j^*}, \infty\big] \cap F_g \mid r_{j^*,m} \le RMC_m(\tau),\ m \mid r_{j^*,m} > 0,\ \tau\in\big[t, t+d_{j^*}\big] \Big\} + d_{j^*}$$

Iteration increment

$g = g+1$

Calculate $A_g(t_g)$, $RMC_m(t_g)$, $E_g = E_g(t_g, Delay_g)$

Update S_g and F_g

$$S_g = S_{g-1} \cup \{ j^* \}$$

$$F_g = F_{g-1} \cup \{ F_{j^*} \}$$

}

}

Calculate Makespan

$F_{n+1} = Max_{l \in P_{n+1}} \{F_l\}$

Figure 5 - Pseudo-code used to construct parameterized active schedules.

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

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 Figure 6). 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 the last two and first two block operations. Otherwise, if $l = 1$ (b) swap only the last (first) two block operations (see Figure 6). 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.

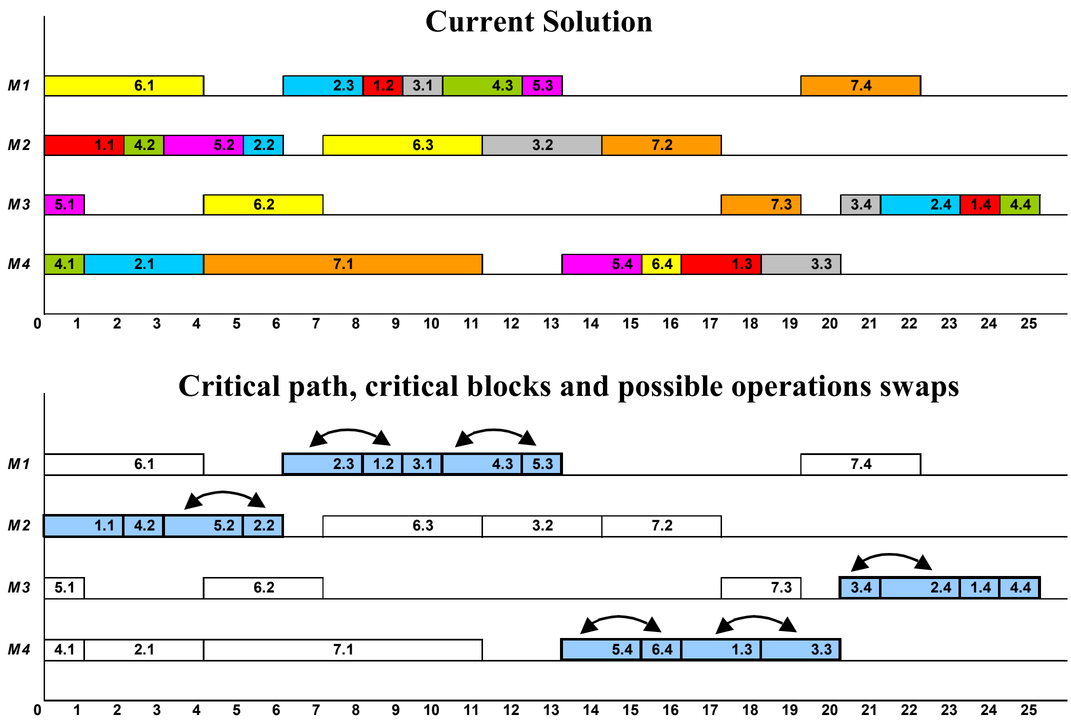

Figure 6 – Neighborhood of Nowicki and Smutnicki (1996).

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 Figure 6.

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
```

Figure 6 – Pseudo-code for 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: Fischer 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 et al. (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 1 summarizes the results. It lists problem name, problem dimension (number of jobs × number of operations), the best known solution (BKS), CPU time (in seconds) for 400 generations of the genetic algorithm, and the solution obtained by each of the algorithms.

Table 1 - Experimental results.

											Dorndorf & Pesch			Aarts et al.		
Instance	Size	BKS	HGA	Time (sec.)	Wang and Zheng (2001)	Aiex et al. (2001)	Binato et al. (2002)	Nowicki and Smutnicki (1996)	Gonçalves and Beirão (1999)	Croce et al.. (1995)	P-GA (1995)	SBGA (40) (1995)	SBGA (60) (1995)	GLS1 (1994)	GLS2 (1994)	Storer et al. (1992)
FT06	6x6	55	55	13	55	55	55	55	55		-					
FT10	10x10	930	930	292	930	930	938	930	936	946	960			935	945	952
FT20	20x5	1165	1165	204	1165	1165	1169	1165	1177	1178	1249			1165	1167	
LA01	10x5	666	666	37	666	666	666	666	666	666	666	666		666	666	666
LA02	10x5	655	655	51		655	655	655	666	666	681	666		668	659	
LA03	10x5	597	597	39		597	604	597	597	666	620	604		613	609	
LA04	10x5	590	590	42		590	590	590	590	-	620	590		599	594	
LA05	10x5	593	593	32		593	593	593	593	-	593	593		593	593	
LA06	15x5	926	926	99	926	926	926	926	926	926	926	926		926	926	
LA07	15x5	890	890	86		890	890	890	890	-	890	890		890	890	
LA08	15x5	863	863	99		863	863	863	863	-	863	863		863	863	
LA09	15x5	951	951	94		951	951	951	951	-	951	951		951	951	
LA10	15x5	958	958	91		958	958	958	958	-	958	958		958	958	
LA11	20x5	1222	1222	197	1222	1222	1222	1222	1222	1222	1222	1222		1222	1222	
LA12	20x5	1039	1039	201		1039	1039	1039	1039	-	1039	1039		1039	1039	
LA13	20x5	1150	1150	189		1150	1150	1150	1150	-	1150	1150		1150	1150	
LA14	20x5	1292	1292	187		1292	1292	1292	1292	-	1292	1292		1292	1292	
LA15	20x5	1207	1207	187		1207	1207	1207	1207	-	1237	1207		1207	1207	
LA16	10x10	945	945	232	945	945	946	945	977	979	1008	961	961	977	977	981
LA17	10x10	784	784	216		784	784	784	787	-	809	787	784	791	791	794
LA18	10x10	848	848	219		848	848	848	848	-	916	848	848	856	858	860
LA19	10x10	842	842	235		842	842	842	857	-	880	863	848	863	859	860
LA20	10x10	902	907	235		902	907	902	910	-	928	911	910	913	916	
LA21	15x10	1046	1046	602	1058	1057	1091	1047	1047	1097	1139	1074	1074	1084	1085	
LA22	15x10	927	935	629		927	960	927	936	-	998	935	936	954	944	
LA23	15x10	1032	1032	594		1032	1032	1032	1032	-	1072	1032	1032	1032	1032	
LA24	15x10	935	953	578		954	978	939	955	-	1014	960	957	970	981	
LA25	15x10	977	986	609		984	1028	977	1004	-	1014	1008	1007	1016	1010	
LA26	20x10	1218	1218	1 388	1218	1218	1271	1218	1218	1231	1278	1219	1218	1240	1236	
LA27	20x10	1235	1256	1 251		1269	1320	1236	1260	-	1378	1272	1269	1308	1300	
LA28	20x10	1216	1232	1 267		1225	1293	1216	1241	-	1327	1240	1241	1281	1265	
LA29	20x10	1157	1196	1 350		1203	1293	1160	1190	-	1336	1204	1210	1290	1260	
LA30	20x10	1355	1355	1 260		1355	1368	1355	1356	-	1411	1355	1355	1402	1386	
LA31	30x10	1784	1784	3 745	1784	1784	1784	1784	1784	1784	-			1784	1784	
LA32	30x10	1850	1850	3 741		1850	1850	1850	1850	-	-			1850	1850	
LA33	30x10	1719	1719	3 637		1719	1719	1719	1719	-	-			1719	1719	
LA34	30x10	1721	1721	3 615		1721	1753	1721	1730	-	-			1737	1730	
LA35	30x10	1888	1888	3 716		1888	1888	1888	1888	-	-			1894	1890	
LA36	15x15	1268	1279	1 826	1292	1287	1334	1268	1305	1305	1373	1317	1317	1324	1311	1305
LA37	15x15	1397	1408	1 860		1410	1457	1407	1441	-	1498	1484	1446	1449	1450	1458
LA38	15x15	1196	1219	1 859		1218	1267	1196	1248	-	1296	1251	1241	1285	1283	1239
LA39	15x15	1233	1246	1 869		1248	1290	1233	1264	-	1351	1282	1277	1279	1279	1258
LA40	15x15	1222	1241	2 185		1244	1259	1229	1252	-	1321	1274	1252	1273	1260	1258

Table 2 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), 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 2 – 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 et al. (1995) - PGA	37	4.61%	0.46%	4.15 %
Dorndorf et al. (1995) - SBGA (40)	35	1.42%	0.48%	0.94 %
Dorndorf et al. (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. (2001)	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.

* AT&T Labs Research Technical Report TD-5EAL6J, September 2002.

References

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

Aiex, R.M., Binato S. and Resende, M.G.C. (2001). Parallel GRASP with Path-Relinking for Job Shop Scheduling, AT&T Labs Research Technical Report, USA. To appear in Parallel Computing.

Adams, J., Balas, E. and Zawack., D. (1988). The shifting bottleneck procedure for job shop scheduling, Management Science, Vol. 34, pp. 391-401.

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

Baker, K.R., (1974). Introduction to Sequencing and Scheduling, John Wiley, New York.

Bean, J.C., (1994). Genetics and Random Keys for Sequencing and Optimization, ORSA Journal on Computing, Vol. 6, pp. 154-160.

Beasley, D., Bull, D.R. and 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. and Resende, M.G.C., (2002). A GRASP for Job Shop Scheduling. In: Essays and Surveys in Metaheuristics, Ribeiro, Celso C., Hansen, Pierre (Eds.), Kluwer Academic Publishers.

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

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

Carlier, J. and Pinson, E., (1990). A practical use of Jackson’s preemptive schedule for solving the job-shop problem. Annals of Operations Research, Vol. 26, pp. 269-287.

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

Croce, F., Menga, G., Tadei, R., Cavalotto, M. and Petri, L., (1993). Cellular Control of Manufacturing Systems, European Journal of Operations Research, Vol. 69, pp. 498-509.

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

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

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

Fisher, H. and Thompson, G.L., (1963). 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, 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.

Garey, M.R. and Johnson, D.S., (1979). Computers and Intractability, W. H. Freeman and Co., San Francisco.

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

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

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

Gonçalves, J.F. and 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. and Hoesada, M. (1991). Matching Heuristic Scheduling Rules for Job Shops to the Business Sales Level, Production and Inventory Management Journal, Vol. 4, pp. 12-17.

Jackson, J.R., (1955). Scheduling a Production Line to Minimize Maximum Tardiness, Research Report 43, Management Science Research Projects, University of California, Los Angeles, USA.

Jain, A.S. and Meeran, S. (1999). A State-of-the-Art Review of Job-Shop Scheduling Techniques. European Journal of Operations Research, Vol. 113, pp. 390-434.

Jain, A. S., Rangaswamy, B. and Meeran, S. (1998). New and Stronger Job-Shop Neighborhoods: A Focus on the Method of Nowicki and Smutnicki (1996), Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland.

Johnson, S.M., (1954). Optimal Two and Three-Stage Production Schedules with Set-Up Times Included, Naval Research Logistics Quarterly, Vol. 1, pp. 61-68.

Laarhoven, P.J.M.V., Aarts, E.H.L. and Lenstra, J.K. (1992). Job shop scheduling by simulated annealing. Operations Research, Vol. 40, pp. 113-125.

Lawrence, S., (1984). Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques, GSIA, Carnegie Mellon University, Pittsburgh, PA.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B., (1993). Sequencing and Scheduling: Algorithms and Complexity in: S.C.Graves, A.H.G. Rinnooy Kari and P.H. Zipkin (eds.), Handbooks in Operations Research and Management Science 4, Logistics of Production and Inventory, Elsevier, Amsterdam.

Lenstra, J.K. and Rinnoy Kan, A.H.G., (1979). Computational complexity of discrete optimisation problems. Annals of Discrete Mathematics, Vol. 4, pp. 121-140.

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

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

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

Perregaard, M. and Clausen, J., (1995). Parallel Branch-and-Bound Methods for the Job_shop Scheduling Problem, Working Paper, University of Copenhagen, Copenhagen, Denmark.

Oliveira, J.A.V. (2000). Aplicação de Modelos e Algoritmos de Investigação Operacional ao Planeamento de Operações em Armazéns, 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: Chrétienne, P., Coffman Jr., E.G., Lenstra, J.K., Liu, Z. (Eds.), Scheduling theory and its application, John Wiley and Sons, pp. 277-293.

Resende, M.G.C., (1997). A GRASP for Job Shop Scheduling, INFORMS Spring Meeting, San Diego, California, USA.

Roy, B. and Sussmann, (1964). Les Problèmes d' ordonnancement avec contraintes dijonctives, Note DS 9 bis, SEMA, Montrouge.

Sabuncuoglu, I., Bayiz, M., (1997). A Beam Search Based Algorithm for the Job Shop Scheduling Problem, Research Report: IEOR-9705, Department of Industrial Engineering, Faculty of Engineering, Bilkent University, Ancara, Turkey.

Spears, W.M. and 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. and Park, I., (1992). Genetic Algorithms in Problem Space for Sequencing Problems, Proceedings of a Joint US-German Conference on Operations Research in Production Planning and Control, pp. 584-597.

Storer, R.H., Wu, S.D., Vaccari, R., (1995). Problem and Heuristic Space Search Strategies for Job Shop Scheduling, ORSA Journal on Computing, 7(4), Fall, pp. 453-467.

Taillard, Eric D. (1994). Parallel Taboo Search Techniques for the Job Shop Scheduling Problem, ORSA Journal on Computing, Vol. 6, No. 2, pp. 108-117.

Vaessens, R.J.M., Aarts, E.H.L. and Lenstra, J.K., (1996). Job Shop Scheduling by local search. INFORMS Journal.

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

Williamson, D. P., Hall, L. A., Hoogeveen, J. A., Hurkens, C. A. J., Lenstra, J. K., Sevast'janov, S. V. and Shmoys, D. B. (1997) Short Shop Schedules, Operations Research, March - April, **45**(2), pp. 288-294.