Genetic algorithms for the job-shop scheduling problem with parallel machines and precedence constraints: heuristic mixing method.

Fatima Ghedjati

LIP6-Université de Paris 6, case 169, 4 Place Jussieu F-75252 Paris cedex 05
tél. (01)44.27.70.10 fax (01)44.27.70.00 e-mail : fatima.ghedjati@lip6.fr

Abstract

In this paper, we are interested in the generalized job-shop scheduling problem with several unrelated parallel machines and precedence constraints (corresponding to linear and non-linear process routings) between the operations of the jobs. The objective is to minimize the maximum completion time (C_{max}). we propose an original resolution method based on genetic algorithms that we call heuristic mixing method, where crossovers mixe a specific heuristics designed for the considered problem. After a description of the problem and of the resolution method, we present experimental results.

Key words: Scheduling, Generalized job-shop, Unrelated parallel machines, Linear and non-linear process routings, Genetic algorithms, Heuristics mixing method.

1. Introduction

Nowadays, the interest of industry, needs effective and fast solving methods, for the resolution of scheduling problems which are referred as NP-hard (Garey & Johnson 1979). In fact, scheduling problems with industrial size, challenge existing exact search methods wich ensure optimality but in an exponential time (see for example Rinnooy Kan 1976, Carlier & Pinson 1989). The application of approximate methods, can't always ensure optimality, but provide near-optimal solutions in a reasonable time. Among these approximate methods, we consider : heuristics methods (for example Baker 1974, French 1982, Ghedjati & Pomerol 1997), AI methods (Rayson 1985, Kusiak & Chen 1988, Fox & Smith 1984), simulated annealing (Aarts & Van Laarhoven 1987), Tabu search method (Glover 1987, Widmer 1991), Genetic algorithms (Goldberg 1989) and neural networks (Hopfield & Tank 1985).
In this paper, we are interested in the factory scheduling problem with several unrelated parallel machines (see for example Horowitz and Sahni 1976, Davis and Jaffe 1981) and precedence contraints between the operations of the jobs. This problem is NP-hard since a simpler problem with two identical machines and with a job restricted to one operation has been shown NP-hard (Van de Velde 1993). This paper proposes an original approach which is the heuristic mixing method based on genetic algorithms, to solve effectively our problem. The next paragraph state the problem at hand and introduce the notations. Paragraph 3 presents the heuristic mixing method, whereas, paragraph 4 reports experimental results.

2. The considered general job-shop problem

The general job-shop factory problem is composed by m machines which have to manufacture n jobs.

- each job has one process routing,

- each process routing j consists of a sequence of k operations $o_{1,j}$, ...,$o_{k,j}$,

- each operation $o_{i,j}$ can be processed by one or several machines: associated with each operation $o_{i,j}$ there is a set M of unrelated parallel machines. The considered operation has to be processed on only one machine r in M during $p_{i,j,r}$ time units without preemption.

We consider on the one hand the sample case of a linear process routing (each job j has to be processed in order to increase indices, i.e. $o_{i+1,j}$ can start only if $o_{i,j}$ has already been completed), and, on the other hand, the more generalized cases which deal with whatever precedence graph, without circuit and not necessary connected between the job operations.

- no machine can process more than one operation at the same time.

- no job can be processed by more than one machine at the same time.

- the problem is static, e.g. all jobs to be make are known and can start at the date zero.

- the goal of scheduling is to minimize the total elapsed time between the beginning of the first operation and the completion of the last operation (the makespan denoted by Cmax).

This problem can be considered as a generalized job-shop problem in two way: on the one hand, it exists several machines which can execute a same operation with different processing time, and, on the other hand, the precedence constraints between the operations doesn't only consider a linear process routing (case of the classical job-shop), but also allows, any non linear process routing. Using the well-known $\alpha/\beta/\gamma$ notation of Graham & al. (1979) and revised by Lawler & al (1982) (see also Blazewicz & al. 1994), we can formulate our generalized job-shop problem as $J(R)/prec/C_{max}$.

example:

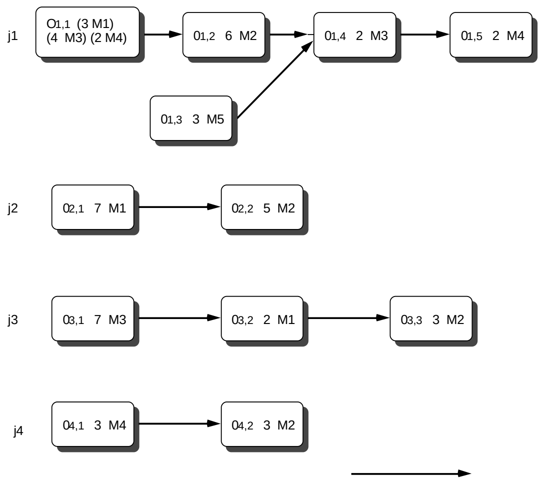

figure 1: Example of a generalized job-shop problem (with unrelated parallel machines and precedence contraints) with 4 jobs and 5 machines

3. Description of the heuristic mixing method

Genetic algorithms (GA) are stochastic search algorithms introduced by John Holland (1975). They parallely explore a large possible solution space of the problem and try to mimic natural evolution processes of an individual population or, in genetic term, of a chromosome population. GA have been extensively studied and applied to a large variety of combinatorial problems, including job-shop problems (Davis 1985, Grefenstette 1987, Biegel & Davern 1990, Bagchi & al. 1991, Syswerda 1991, Fang & al. 1993, Uckun & al. 1993, Portmann & Ghedjati 1995, Ghedjati 1996 ...). The heuristic mixing method which we designed and developed, is an original approach which type is "heuristic space" that is defined in (Storer & al. 1992) and utilized also by Chiu & Yih (1995) and Dorndorf et Pesch (1995). In our approach, we utilize the GA to give to a scheduler builder (see Ghedjati 1994, Ghedjati & Pomerol 1997), on the one hand, a series of different priority rules for the choice of operations (among the list of the operations to be affected at a given machine), and, on the other hand, a series of heuristic rules for the choice of the machine when a given operation has a multiple choice of machines. The originality is how we combine both of these technics together in the same GA.

The principal steps of the genetic algorithm we use are as follows:

1. creation of initial population;
2. crossover (to perform a given number of mating);
3. selection (to conserve a limited number of individus);
4. mutation (used with probability 0.5);
5. if a fixed number of steps is reached stop; if not go to 2.

3.1. Coding

Our method is based on an indirect domain-independent approach (see for example, Falkenauer & Bouffouix 1991, Nakano & Yamada 1991, Starkweather & al. 1991, Whitley et al. 1991, Bruns 1992). In a such approache, a solution is not directly represented in the chromosome as it is in a direct representation case (see foe example Kanet & Sridharan 1991, Bruns 1993, Djerid 1997). An indirect domain-independant representation and an indirect problem-specific representation (Davis 1985, Bagchi & al. 1991, Uckun & al. 1993, Ghedjati 1996), are not similar but they are both characterized by the incorporation in the chromosome of parameters which allow the schedule builder to build a solution of the problem. In our system (see Ghedjati 1994, Ghedjati & Pomerol 1997), the schedule builder proceeds to the choice of the current machine. On this machine, some operations are candidate for an assignment. In our coding (see example 1), the second line gives the priority rule to be applied for the choice of the operation with the highest priority. The third line determines on which machine the operation which is chosen at the precedent step (if it has a multiple choice of machines) will be effectively assigned by the application of machine choice heuristics.

So, in our coding, a chromosome consists of two parts.

- The first part gives, for each corresponding gene, so to say, for each potential assignment of operation, a number of priority rule which allows the choice of the operation with highest priority (among the list of the operations which are candidate for an assignment). The rules are numbered from 1 to nb (nb is the total number of the utilized rule). For example, if this number is **1**, the priority rule is SPT: "Shortest processing time"; if it is **2**, the operation is randomly chosen. We can also utilize any priority rule which is adapted to this problem (see for example Panwalkar & Iskandar 1977).

- The second part gives, for each potential assignment of operation, a number corresponding to one of the 7 heuristics (H1 to H7), (see Ghedjati & Pomerol 1997). This number allows to compute dynamically, when we want to assign a new operation (which has a multiple choice of machines), to which machine we must affect it, depending on whether this number is equal to 1, 2, 3, ..., or 7. For example, **1** is corresponding to the heuristic H1: The operation is affected to the first available machine; **2** is corresponding to the heuristic H2: high priority is given to the faster machine; **3** is corresponding to the heuristic H3: high priority is given to the less loaded machine (the load of the machine is computed dynamically the further the solution is being built); **4** is corresponding to the heuristic H4 which uses ″probabilities″. The highest priority operation is affected to the most desirable machine which is obtained by drawing lots according to ″probabilities″. This ″probabilities″ balance the load of the machines and are dynamically computed wherever a machine is chosen. H4 permits to get random choices, it may be executed many times and so it provides different results. H4(nb) consists in using nb times H4 and in keeping the best result. We should use H4(1) so that its processing time may be compared with the processing time of the other heuristics. **5** is corresponding to the heuristic H5, which is very close to H4, however, instead of drawing lots, we use computed ″probabilities″ as priority rules in order to make choices; **6** (**7** respectively) is corresponding to the heuristic H6 (H7 respectively) variant of the heuristic H3 (H5 respectively).

Example 1: the considered examples in this paper, correspond to the example in figure 1 where the operation j of the part i is represented by ij.

parent1

operation	11	12	13	14	15	21	22	31	32	33	41	42
priority rule(choice of operation)	2	1	2	1	2	2	2	1	2	1	2	2
heuristic for the choice of machine	2	4	3	2	2	2	5	2	3	3	1	2

3.2. Crossover

Generally, the scheduling problems needs specific genetic operators. However, in an indirect domain-independent representation, the chromosome coding does not contain information about the problem domain. So, the usual genetic operators which are applied to different problems can be used in this case.

We use one point crossover which is designed for classical GA ($child_1$ is obtained with the beginning of parent1 and the end of parent2, we parallely obtain the $child_2$ chromosome by inversing the parents role). All traditional crossovers defined for the bit string representation can be easily used in this case. The crossover allows so, to mix and to apply different heuristic rules (instead of applying the same heuristic rule for all the operations.

Example 2: We suppose that the randomly chosen cut point is in position 8.

parent2

operation	11	12	13	14	15	21	22	31	32	33	41	42
priority rule(choice of operation)	1	2	1	2	2	2	1	2	2	1	2	1
heuristic for the choice of machine	5	4	2	4	3	3	1	1	1	4	5	4

$child_1$

operation	11	12	13	14	15	21	22	31	32	33	41	42
priority rule(choice of operation)	2	1	2	1	2	2	2	1	2	1	2	1
heuristic for the choice of machine	2	4	3	2	2	2	5	2	1	4	5	4

Then, we use the content of the children chromosome to compute the solution. As a matter of fact, the content of a chromosome gives information to the scheduler builder, on which priority rule is applied to a given operation, to build a solution. We insert this individual (if it does not already exist in the population) in order to increase Cmax; and we permanently keep in memory the best solution found so far.

3.3. Mutation

A chromosome may mutate, thus generating a new individual which is inserted in the population if it does not already exist. The chances for mutation are the same than the number of mating during the reproduction phase [if we have to apply a mutation with a certain probability, the number of mutations (NBM) to be performed will be incremented from 1 to a fixed number of mating (NBT) or until we generate a new chromosome (which does not already exist) in the population]. These steps are described as follows:

1. NBM = 0;
2. mutation;
3. compute the new chromosome solution;
4. if the mutant doesn't already exist, insert it in order to increase C_{max}; update the best C_{max} of the population; go to 7;
5. if the mutant already exists in the population, NBM = NBM + 1;
6. if NBM < NBT go to 2;
7. end.

We utilized a domain-independent mutation. We randomly select two genes and the corresponding values are swapped for both parts of the coding.

3.4. Strategies linked to mutations and crossovers

We have designed and developed a non usual variety of GA. We suppose that a bad mutant would not be selected for a crossover in the next step and disappears without giving the wished diversity to the population. We use strategies which can force or not the new mutants to procreate at the next iteration, in order to verify if this possibility improves or not the best obtained result.
We use 4 strategies when selecting the parents:

(S1) The mutant is not obliged to procreate and the parents are randomly selected with equal probability among the N best genitors (N is the size of the population).

(S2) The mutant have to procreate and all others parents are randomly selected with equal probability among the N best individuals.

(S3) The mutant is not obliged to procreate and the parents are selected with the roulette technique (see for example Davis 1991): parents are randomly selected, but with a probability increasing with the best individuals.

(S4) The mutant have to procreate and the parents are selected with the roulette technique.

3.5. Selection

We use two selection techniques : one is a stochastic selection which keeps a fixed number (q) of individuals selected randomly with the roulette technique, and the second is a deterministic selection which keeps the (q) best individuals of the population.

4. Experimental results

We have implemented this work in a C/Unix/SUN-Sparc10 environment, and tested it on different benchmarks; one of them is coming from the literature; the others are randomly generated.
We reported the following characteristics of experiments:

m x n: problem with m machines and n jobs;
t: CPU time expressed in seconds;
SP small processing time;
LP: large processing time;
mx : application of the heuristic mixing method
Si: strategy number i;
mx(Si) application of the heuristic mixing method with strategy number i;
AS: all strategies;
Avr. GA: average of the obtained best solutions by applying the 4 GA strategies.
AS gain in % / heuristics: gain in % of all the strategies in comparison with heuristic solutions;

4.1. The R//C_{max} problem

We consider first, a literature problem which is simpler than ours. It’s the unrelated parallel machine scheduling problem, where a job is restricted to one operation, and there is no precedence constraints. We consider the (3 x 8) example of Van de Velde (1993). The results are ported in the following table

Example	method	3 x 8 C_{max}	t
Van de Velde 1993	Lagrangian relaxation	20	_
our heuristics	H1	27	39374 µs
"	H2	33	49441 µs
"	H3	36	76493 µs
"	H4(1)	24	1.5 s
"	H4(5)	22	33.5 s
"	H4(10)	22	72 s
"	H5	25	3 s
"	H6	25	6 s
"	H7	25	5s et 63660 µs
heuristics mixing method	mx(S1), mx(S2), mx(S3), mx(S4)	20	5 s

table 1

The best solution stemmed from the heuristics is given by the heuristic H4 when it is executed 5 times. It is near 10% from the optimal solution (20).

Improvement of the heuristics results with the GA or with a mixing heuristics method:

We start with an initial population size of 20, that is increased to 50. The number of generation is 50. the initial population is obtained by mixing heuristics solutions (stemmed from the heuristics H1,...,H7), and the randomly generated solutions.
The optimum solution (see table 1) is quickly obtained for all the strategies.

4.2. Randomly generated benchmarks

We also tested this method on 12 randomly generated examples of our problem and which have totally different process routing; for details see (Ghedjati 1994, Ghedjati&Pomerol 1997).
For the 12 examples, we have realized until now 3 series of experiments with population size of 10mx, 20mx and 30mx (see notation). We start with 10 individuals for the first series and with 20 individuals for the remainder (which increases until 30 in the third series). For each series, we have applied our 4 strategies (Si). We first report the global results and then the results taking into account the processing times of the operations (small processing time [1,5] or large one [1,50]). The initial population is stemmed by mixing heuristics solutions and randomly generated solutions. The number of generations is 50 and the number of mating (for each generation) is 5. We have tested both stochastic and deterministic selection techniques, but after the experiments, we considered only the second one (the first one gives bad results).
We can say about the results in table 2 that :
- no strategy is systematically better than another (according to these examples, forcing the mutants to systematically procreate or not doesn't induce a significant difference on the average results);
- applying successively all strategies and taking the best solution gives a better gains than executing only one strategy.

series	10mx	20mx	30mx
S1 gain in % / heuristics	9.20	8.11	11.16
S2 gain in % / heuristics	9.07	8.24	9.41
S3 gain in % / heuristics	8.55	8.24	10.72
S4 gain in % / heuristics	9.29	8.11	10.11
TS gain in % / heuristics	11.04	9.75	11.77

table 2

In table 3, the given mean represents the total duration of the best schedule (of the considered sample examples) built according to the diversity of operations processing times and after applying the 4 GA strategies (for each term of the specimen examples, we carry the best solution of the 4 strategies).

Avr. GA	SP	LP	SP ∪ LP
30mx	**36.16**	**301.33**	**168.75**
20mx	36.34	303.17	169.75
10mx	36.83	301.50	169.17

table 3

We note (on the experiments already realized) that the best population size is 30. At the present time, others experiments with more important size of population are conducted.

Processing time

The average processing time of creation of the heuristics solutions to construct the first part of the initial population is 2,36 seconds. The average processing time of creation of the total initial population (heuristic solutions and randomly generated solutions) and one GA strategy is de 261 seconds.

In conclusion, the heuristic mixing method based on GA allows to obtain up to 10% of improvement according to the heuristics, but with a longer processing time.

5. Conclusion

We have proposed an heuristic mixing method based on GA to solve generalized job-shop scheduling problems (with unrelated parallel machines and precedence constraints). We have designed and developed non usual methodologies which enrich this method. They concerned the number of the mutations carried out in order to succeed a mutation, as well as the strategies linked to the genetic operators, which force, if we wish, the new mutants to procreate at the next iteration, in order to avoid the establishment of uniform populations. The results obtained with the random benchmarks we have generated so far, show that the best population size is 30 for 50 generations, whatever the considered operations processing time may be. The improvement due to GA on the considered series of experiments, is up to 10% of the average in comparison with the heuristic solutions, but in a longer running time. At present, others experiments are conducted, in particular on a largest population size and on a more important number of generations, in order to analyze the performance evolution of the population according to the parameters. Others literature benchmarks will also be considered. In future research, we will consider this idea, and make the comparison between our results and results obtained by simulated annealing or tabu search method.

References

E.H.L. Aarts, P.J.M. Van Laarhoven (1987), Simulated Annealing : Theory and Applications. D. Reidel Publishing Company, Dordrecht, Holland.

K.R. Baker (1974), Introduction to sequencing and scheduling. John Wiley and Sons, New-York.

S. Bagchi, S. Uckun, Y. Miyabe, K. Kawamura (1991), Exploring problem-specific recombination operators for job-shop scheduling. Proceedings of the Fourth International Conference on Genetic Algorithms and their Applications, 10-17.

J.E. Biegel, J.J. Davern (1990), Genetic algorithms and job-shop scheduling, Computers ind. Engng Vol. 19, Nos 1-4, 81-91.

J. Blazewicz, K.H. Ecker, G. Schmidt, J. Weglarz (1994), Scheduling in Computer and Manufacturing Systems. Second, Revised Edition, Springer-Verlag.

R. Bruns (1992), Incorporation of a knowledge-based scheduling system into a genetic algorithm. Proceedings of the GI-Jahrestagung, Springer.

R. Bruns (1993), Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling. Proceedings of the Fifth International Conference on Genetic Algorithms and their Applications, 352-359.

J. Carlier, E. Pinson (1989), An algorithm for solving the job-shop problem. Man. Sci., 35, 164-176.

C. Chiu, Y. Yih (1995), The learning-based methodology for dynamic scheduling in distributed manufacturing system. Int. J. of Prod. Res., to appear.

E. Davis, J.M. Jaffe (1981), Algorithms for scheduling task on unrelated processors. Journal of the Association for Computing Machinery, 28(4): 721-736.

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

L. Davis (1991), Handbook of genetic algorithms. Van Nostrand Reinhold, New York.

L. Djerid (1997), Hybridation d'algorithmes génétiques et de méthodes classiques de recherche opérationnelle pour résoudre des problèmes d'ordonnancement. Thèse de l'Institut National Polytechnique de Lorraine.

U. Dorndorf, E. Pesch (1995), Evolution based learning in a job-shop scheduling environment. Computer. Oper. Res., to appear.

E. Falkenauer, S. Bouffouix (1991), A genetic Algorithm for job-shop, Proceedings of the 1991 IEEE, International Conference on Robotics and Automatisation, Sacramento, California.

H.L. Fang, P. Ross, D. Corne (1993), A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. Proceedings of the Fifth International Conference on Genetic Algorithms and their Applications, 375-382.

M.S. Fox, S.F. Smith (1984), ISIS: a knowledge based system for factory scheduling. Expert Systems Journal, Vol. 1, n°1, 25-45.

S. French (1982), Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop. Wiley.

M.R. Garey, D.S. Johnson (1979), Computers and Interactability: A Guide to the Theory of NP-completeness, Freeman and Co.

F. Ghedjati (1994), Résolution par des heuristiques dynamiques et des algorithmes génétiques du problème d'ordonnancement de type job-shop généralisé (à machines non identiques en parallèle et contraintes de précédence). Thèse de l'Université Paris 6.

F. Ghedjati (1996), Genetic algorithm for the generalized job-shop scheduling problem. IPMU'96 (Information Processing and Management of Uncertainty in Knowledge-Based Systems), Granada (Spain), July 1-5, Vol. 2, 793-801.

F. Ghedjati, J.C. Pomerol (1997), Résolution du problème d'ordonnancement d'ateliers de type job-shop généralisé par des heuristiques dynamiques. Rapport du LIP6 1997/005, Juin. Soumis à la revue JESA.

F. Glover (1987), Tabu search methods in artificial intelligence and operations research. ORSA, Artificial Intelligence Newletter, 1, no. 2.

D.E. Goldberg (1989), Genetic Algorithms in Search, Optimisation, and Machine Learning. Addison-Wesley Publishing Company, Inc.

R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1979), Optimisation and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics, 5, 287-326.

J.J. Grefenstette (1987), Incorporating problem-specific knowledge into genetic algorithms, in Genetic Algorithms and Simulated Annealing, L. Davis, ed., Morgan Kaufman, San Mateo, Calif., 42-60.

J. Hopfield, J. Tank (1985), Neural computation of decisions in optimisation problems. Biological Cybernetics, 52, 141-152.

E. Horowitz, S. Sahni (1976), Exact and approximate algorithms for scheduling nonidentical processors. Journal of the Association for Computing Machinery, 23(2): 317-327.

J.J. Kanet, V. Sridharan (1991), PROGENITOR : A genetic algorithm for production scheduling. Wirtschaftsinformatic.

A. Kusiak, M. Chen (1988), Expert systems for planning and scheduling manufacturing systems. E. J. of Operational Research, 34, 113-130.

E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1982). Recent developments in deterministic sequencing and scheduling: a survey. In M.A.H. Dempster, J.K. Lenstra, and A.H.G. Rinnooy Kan, editors, Deterministic and stochastic scheduling, 35-73, Dordrecht, The Netherlands, 1982. NATO Advanced Study and Research Institute, D. Reidel Publishing Company.

R. Nakano, T. Yamada (1991), Conventional genetic algorithm for job-shop problems. Proceedings of the Fourth International Conference on Genetic Algorithms and their Applications, 475-479.

S.S. Panwalkar, W. Iskandar (1977), A Survey of Scheduling Rules. Operations Research, 25 (1): 45-61.

M. C. Portmann, F. Ghedjati (1995), Affectation et Ordonnancement par des Algorithmes Génétiques. Journées d'Etudes "Affectation et Ordonnancement", GDR Automatique du CNRS (GT3 Ordonnancement), EIII TOURS, 18-19 septembre, 67-80.

P. Rayson (1985), A review of expert systems principales and their roles in manufacturing. Robotica, vol. 3.

A.H.G. Rinnooy Kan (1976), Machine Scheduling Problems: classification, complexity and computation. Nijhoff, The Hague.

T. Starkweather, S. McDaniel, K. Mathias, D. Whitley (1991), A comparison of genetic sequencing operators. Proceedings of the Fourth International Conference on Genetic Algorithms and their Applications, 69-76.

R.H. Storer, S.Y.D. Wu, Vaccari (1992), New Search Spaces for Sequencing Problems With Application to Job Shop Scheduling. Management Science, 38 (10), 1495-1509.

G. Syswerda (1991), Schedule Optimization using Genetic Algorithms. In Handbook of Genetic Algorithms, L. Davis, ed., Van Norstrand Reinhold, New York, 332-349.

S. Uckun, S. Bagchi, K. Kawamura (1993), Managing genetic search in job-shop scheduling. IEEE Expert, 8(5), 15-24

S.L. Van de Velde (1993), Duality-based algorithms for scheduling unrelated parallel machines. ORSA Journal on Computing, vol. 5, No. 2, 192-205.

D. Whitley, T. Starkweather, D. Shaner (1991), The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination. In [Davis 91], 350-372.

M. Widmer (1991), Job-shop scheduling with tooling constraints: a Tabu Search Approach, J. Opl Res. Soc. Vol.42, No. 1, 75-82.