Scheduling Jobs in Flexible Manufacturing Cells with Genetic Algorithms

António Ferrolho[1] and Manuel Crisóstomo[2]

[1] Superior School of Technology of Viseu, Polytechnic Institute of Viseu
Campus Politécnico de Repeses, 3504-510 Viseu, Portugal
antferrolho@elect.estv.ipv.pt
[2] Institute of Systems and Robotics, University of Coimbra
Polo II, 3030-290 Coimbra, Portugal
mcris@isr.uc.pt

Abstract. In this paper, we studied scheduling problems with Genetic Algorithms (GA) in Flexible Manufacturing Cells (FMC). We used a GA for solving the optimization scheduling problem. First, we developed an FMC with industrial characteristics, with the objective of studying scheduling problems in these types of manufacturing systems. Then, we developed a software tool, called HybFlexGA, to study scheduling problems, with GA, in the FMC. Finally, we applied the HybFlexGA to solve scheduling problems in the FMC. The practical results obtained from the FMC for the various scheduling problems show how efficient HybFlexGA is in solving these problems.

Keywords: Flexible manufacturing cells, scheduling and genetic algorithms.

1 Introduction

Scheduling problems in Flexible Manufacturing Cells (FMC) are studied in this paper. We also use Genetic Algorithms (GA) to optimize this type of problems. We have developed an FMC with industrial characteristics and a software tool, called HybFlexGA, with the objective of studying scheduling problems in this type of manufacturing systems. The FMC and the HybFlexGA were used to study single machine total weighted tardiness (SMTWT) problems.

In SMTWT problems each job i has an associated processing time p_i, a weight w_i, and a due date d_i, and the job becomes available for processing at time zero. The tardiness of a job i is defined as T_i=max{0, C_i-d_i}, where C_i is the completion time of job i in the current job sequence. The objective is to find a job sequence which minimizes the sum of the weighted tardiness given by $\sum_{i=1}^{n} w_i.T_i$. Because the SMTWT problem is NP-hard, optimal solutions for this problem would require a computational time that increases exponentially with the problem size [1] and [2]. In recent years, several heuristics, such as Simulated Annealing, Tabu Search, Genetic Algorithms and Ant Colony [1] and [3], have been proposed to solve the SMTWT problem.

Advances in Computer Science and Engineering.
Research in Computing Science 23, 2006, pp. 31-40

Received 04/07/06
Accepted 03/10/06
Final version 13/10/06

2 Developed Flexible Manufacturing Cell

An FMC with industrial characteristics was developed with the objective of studying scheduling problems in these types of manufacturing systems. The hierarchical structure implemented in the FMC is shown in Fig. 1. This FMC is comprised of four sectors, which are controlled by PCs and different software [5] and [6]. The four sectors are:

- The manufacturing sector, made up of two CNC machines (mill and lathe), one ABB IRB140 robot and one buffer (see Fig. 1);
- The assembly sector, made up of one Scorbot ER VII robot, one small conveyor and an assembly table (see Fig. 1);
- The handling sector, made up of one big conveyor (see Fig. 1);
- The storage sector, made up of five warehouses and one robot ABB IRB1400 (see Fig. 1).

Control of existing equipment in each sector is carried out by four computers: PC1 – manufacturing sector, PC2 – assembly sector, PC3 – handling sector and PC4 – storage sector. Coordination, synchronization and integration of the four sectors is carried out by the of FMC Manager computer.

The first layer contains the engineering and design functions where the product is designed and developed. The outputs of this activity are the drawings and the bill of materials.

The second layer is process planning. Here the process plans for manufacturing, assembling and testing are carried out.

The third layer is scheduling. The process plans together with the drawing, the bill of materials and the customer orders are the input to scheduling. The output of scheduling is the release of the order to the manufacturing floor. The PCM – FMC Manager is responsible for the engineering, planning and scheduling activities.

The fourth layer is control. Manufacturing is controlled by a hierarchically structured real time computer system (PC1, PC2, PC3 and PC4). Theirs set points are the operating parameters used to start and control the activities on the production floor.

The fifth layer is data acquisition. The operations of the machine tools and material movement equipment are monitored by a data acquisition system. The collected data represents the state of the manufacturing system and is the feedback information used for control.

The central computer (FMC Manager) controls all of the production of the FMC, calling several computers and using local nets to exchange data which allow control and supervision of the operations in real time, picking up and processing information flows from the various resources. Concisely, the FMC manager PC is responsible for:

- Developing and designing new products to manufacture – the engineering layer;
- Production plans, assemblies and product tests – the planning layer;
- Finding the optimum processing sequence so as to optimize CNC machine use – the scheduling layer;
- Coordination, integration and control of all the sectors in the FMC;

- Maintaining a database of jobs to manufacture, including the respective NC programmes;
- Synchronizing the various sectors so as to produce variable lots of different types of parts depending on the customer's orders;
- Monitoring the current state of production;
- Guaranteeing tolerance of failures, safety and coherence of data.

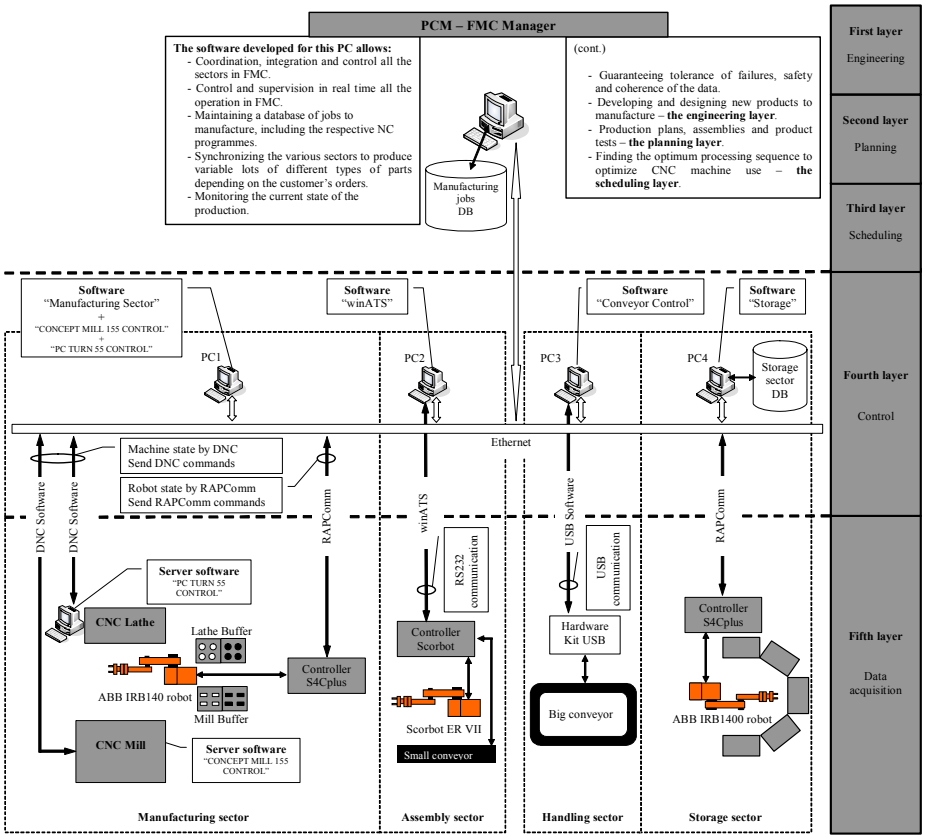


Fig. 1. FMC hierarchical structure.

3 Hybrid and Flexible Genetic Algorithm

We developed a software tool, called HybFlexGA (Hybrid and Flexible Genetic Algorithm), to solve scheduling problems in developed FMC (see the third layer in Fig. 1). The HybFlexGA was coded in C++ language and its architecture is composed of three modules: interface, pre-processing and scheduling module.

3.1 Interface Module

The interface module with the user is very important for the scheduling system's success. Thus, this interface should be user-friendly and dynamic so as to allow easy manipulation of the scheduling plan, jobs, and so forth. This interface allows the connection between the user and the scheduling module, facilitating data entry (for example, parameter definition and problem definition) and the visualization of the solutions for the scheduling module. Fig. 2 shows the interface window.

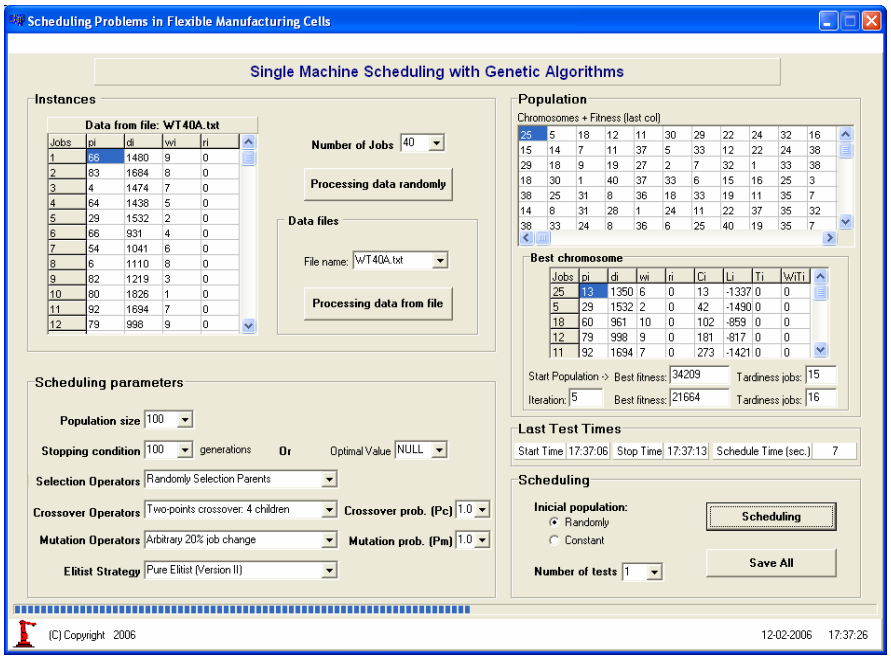


Fig. 2. Interface window.

3.2 Pre-processing Module

The inputs of the pre-processing module are the problem type and the scheduling parameters. The instance of the scheduling problem can be randomly generated or generated by PC file, as shown in Fig. 2. This module pre-processes the input information and then sends the data to the next module – the scheduling module.

3.3 Scheduling Module

The objective of the scheduling module is to give the optimal solution for any scheduling problem. If the optimal solution is not found, the GA gives the best solution found (near-optimal solution). In this module, we implemented the GA shown in Fig. 3.

Step 1 - Initialization
Let t=0, where t is the generation index, and generate an initial population randomly Ψ_0 including N_{pop} solutions (N_{pop} is the number of solutions in each population, i.e., N_{pop} is the population size). The number of solutions (chromosomes) in the t generation is given by $\Psi_t = \left\{x_t^1, x_t^2, ..., x_t^{N_{pop}}\right\}$.

Step 2 - Selection
Select pairs of solutions (parents' chromosomes) from the current population Ψ_t. Each chromosome x_t^i is selected according to the selected operator chosen in the interface module.

Step 3 - Crossover
Apply a crossover operator, selected in the interface module, to each of the selected pairs in step 2. This way, new chromosomes will be generated according to the selected crossover probability (P_c).

Step 4 - Mutation
Apply a mutation operator, selected in the interface module, to the generated chromosomes in step 3, according to the selected mutation probability (P_m).

Step 5 – Elitism
Select the best N_{pop} chromosomes to generate the next population Ψ_{t+1} and the other chromosomes are eliminated. Thus, the best chromosomes, i.e. solutions, will survive into the next generation. However, duplicated solutions may occur in Ψ_{t+1}. To minimize this, new chromosomes are generated for all duplicated chromosomes.

Step 6 – Termination test
Stops the algorithm if the stopping condition, previously specified in the interface module, is satisfied. Otherwise, update t for t:=t+1 and return to step 2.

Fig. 3. GA implemented in the scheduling module.

4 Genetic Operators

In this section, we propose a new concept of genetic operators for scheduling problems. We evaluate each of various genetic operators with the objective of selecting the best performance crossover and mutation operators.

4.1 Crossover Operators

Crossover is an operation to generate a new sequence from two sequences. We examine the following crossover operators:

- One-point crossover: 1 child (OPC1C) in Fig. 4 a);
- Two-point crossover: 1 child (Version I) (TPC1CV1) in Fig. 4 b);
- Two-point crossover: 1 child (Version II) (TPC1CV2) in Fig. 4 c).

We also developed crossover operators with 2, 3 and 4 children. The crossover operators with 2 children are:

- One-point crossover: 2 children (OPC2C). This crossover is similar to the OPC1C, but it generates two children;

- Two-point crossover: 2 children (Vers. I) (TPC2CV1). This crossover is similar to the TPC1CV1, but it generates two children;
- Two-point crossover: 2 children (Vers. II) (TPC2CV2). This crossover is similar to the TPC1CV2, but it generates two children.

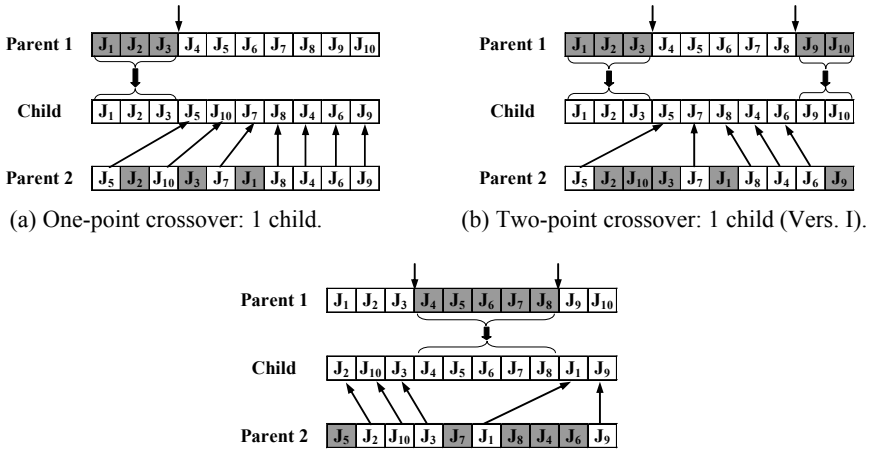
The crossover operators with 3 children are:

- Two-point crossover: 3 children (Version I) (TPC3CV1). This crossover is a mix of TPC1CV1 plus TPC2CV1;
- Two-point crossover: 3 children (Version II) (TPC3CV2). This crossover is a mix of TPC1CV2 plus TPC2CV2.

The crossover operator with 4 children is called a two-point crossover: 4 children (TPC4C). This operator is a mix of TPC2CV1 plus TPC2CV2.

The following three crossover operators are also examined in this paper for FMC scheduling problems:

- Order crossover (OX) in Goldberg [4];
- Cycle crossover (CX) in Oliver [7];
- Position based crossover (PBX) in Syswerda [8].



(a) One-point crossover: 1 child.

(b) Two-point crossover: 1 child (Vers. I).

(c) Two-point crossover: 1 child (Vers. II).

Fig. 4. Illustration of crossover operators.

4.2 Mutation Operators

We examined the following four mutations used by Murata in [9]: adjacent two-job change (Adj2JC), arbitrary two-job change (Arb2JC), arbitrary three-job change (Arb3JC) and shift change (SC).

We developed a new mutation operator called the arbitrary 20%-job change (Arb20%JC), as we can see in Fig. 5. This mutation selects 20% of the jobs in the child chromosome. The 20% of the jobs to be changed are arbitrarily and randomly selected, and the order of the selected jobs after the mutation is randomly specified.

The percentage in this mutation operator gives the operator some flexibility, i.e. the number of jobs to be changed depends on the size of the chromosome.

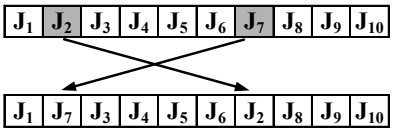


Fig. 5. Arbitrary 20%-job change.

4.3 Examination of Crossover and Mutation Operators

The aim of this subsection is to examine the twelve crossover operators and the five mutation operators, presented in the last two subsections.

When the crossover operators were examined the mutation operator was not used and when the mutation operators were examined the crossover operator was not used. Each crossover operator was examined by using the following conditions: number of tests, 20; initial population Ψ_t, constant; number of jobs, 40; instance used, constant; population size N_{pop}, 20, 40, 60, 80 and 100; stopping condition, 1000 generations; crossover probabilities P_c, 0.2, 0.4, 0.6, 0.8 and 1.0; mutation probabilities P_m, 0.2, 0.4, 0.6, 0.8 and 1.0; mutation operators and mutation probabilities, not used in the examination of crossover operators; crossover operators and crossover probabilities, not used in the examination of mutation operators.

We used the following performance measure with the aim of evaluating each genetic operator:

$$Performance = f(\bar{x}_{initial}) - f(\bar{x}_{end}) \quad \textbf{(1)}$$

where $x_{initial}$ is the best chromosome in the initial population and x_{end} is the best chromosome in the last population. That is, $f(\bar{x}_{initial})$ is the fitness average (of the 20 computational tests) of the best chromosomes in the initial population and $f(\bar{x}_{end})$ is the fitness average of the best chromosomes at the end of the 1000 generations. The performance measure in (1) gives the total improvement in fitness during the execution of the genetic algorithm.

We used 20 computational tests to examine each crossover and mutation operator. The average value of the performance measure in (1) was calculated for each crossover and mutation operator with each crossover probability (P_c), each mutation probability (P_m) and each population size (N_{pop}). Table 1 and Table 2 show the best average value of the performance measure obtained by each crossover operator and by each mutation operator with its best crossover probability, best mutation probability and best population size.

Table 1. Classification of the crossover operators.

Position	Crossover	P_c	N_{pop}	Performance
1st	TPC4C	1.0	100	3834.1
2nd	TPC3CV2	1.0	100	3822.9
3rd	TPC2CV2	1.0	100	3821.8
4th	PBX	1.0	80	3805.8
5th	TPC1CV2	0.8	100	3789.3
6th	CX	0.8	80	3788.7
7th	TPC3CV1	0.8	80	3680.2
8th	TPC2CV1	1.0	80	3662.1
9th	OPC2C	0.6	100	3647.8
10th	OX	1.0	100	3635.4
11th	TPC1CV1	1.0	100	3624.7
12th	OPC1C	0.6	100	3570.5

Table 2. Classification of the mutation operators.

Position	Mutation	P_m	N_{pop}	Performance
1st	Arb20%JC	1.0	100	3833.9
2nd	Arb2JC	0.8	100	3826.4
3rd	Arb3JC	1.0	60	3814.9
4th	SC	0.8	60	3673.5
5th	Adj2JC	0.4	100	3250.4

5 Computational Tests

This section presents the computational results obtained with 40, 50 and 100 jobs. From the OR-Library [10] we randomly selected some instances of SMTWT problems with 40, 50 and 100 jobs. We used 20 computational tests for each instance of the SMTWT problem. We used the six best crossover operators (see Table 1) and the best mutation operator (see Table 2) in the HybFlexGA. Each instance of the SMTWT problem was examined using the following conditions:

- Number of tests: 20;
- Initial population Ψ_t : randomly generated;
- Number of jobs: 40, 50 and 100;
- Instance used: from the OR-Library [10];
- Population size N_{pop}: 80 and 100 (see Table 1 and Table 2);
- Stopping condition: 1000 generations for the instances with 40 and 50 jobs or the optimal solution, and 5000 generations for the instances with 100 jobs or the optimal solution;
- Crossover operators: the six best crossover operators in Table 1;
- Crossover probabilities P_c: 0.8 and 1.0 (see Table 1);
- Mutation operators: the best mutation operator in Table 2;
- Mutation probabilities P_m: 1.0 (see Table 2).

Table 3 shows the computational results obtained for the SMTWT problems with 40, 50 and 100 jobs. In this table we have the number of tests with optimal solution, the CPU time average (in seconds) and the generation average for each instance problem. For example, we chose the TPC4C with P_c=1.0, Arb20%JC with P_m=1.0 and instance 40A (SMTWT problem with 40 jobs, from the OR-Library [10]) in the HybFlexGA. We used 20 computational tests for this instance. In these tests we obtained the optimal solutions in 16 tests. In these 16 tests, the CPU time average was 362.4 seconds and the generation average was 593.

As shown in Table 3, we obtained good results with the TPC4C+Arb20%JC, TPC3CV2+Arb20%JC and TPC2CV2+Arb20%JC combination, for all the instances with 40, 50 and 100 jobs. However, this table also shows the best results are obtained for the TPC4C+Arb20%JC combination.

When we used the TPC4C+Arb20%JC combination, the HybFlexGA is very efficient. For example, in the six instances with 40, 50 and 100 jobs (see Table 3) the HybFlexGA found 20 tests with an optimal solution in four instances (40B, 50A, 50B and 100B), and 16 tests with optimal solutions in two instances (40A and 100A).

Table 3. Computational results obtained for the SMTWT problems with 40, 50 and 100 jobs.

	Instance	**40A**	**40B**	**50A**	**50B**	**100A**	**100B**
	Optimal solution	**6575**	**1225**	**2134**	**22**	**5988**	**8**
TPC4C + Arb20%JC	Tests with optimal solution	16	20	20	20	16	20
	CPU time average (sec.)	362.4	190.0	88.3	45.5	2405.1	523.9
	Generations average	593	284	107	54	1611	323
TPC3CV2 + Arb20%JC	Tests with optimal solution	13	15	18	20	15	20
	CPU time average (sec.)	382.9	231.3	112.3	50.4	3851.0	1012.4
	Generations average	725	402	158	70	3042	727
TPC2CV2 + Arb20%JC	Tests with optimal solution	8	16	17	20	9	20
	CPU time average (sec.)	369.1	216.8	146.2	92.0	3921.3	1339.8
	Generations average	380	449	246	152	3685	1142
PBX + Arb20%JC	Tests with optimal solution	0	8	18	20	1	20
	CPU time average (sec.)	----	236.1	144.6	86.3	2067.0	1413.6
	Generations average	----	747	371	218	2957	1828
TPC1CV2 + Arb20%JC	Tests with optimal solution	0	3	13	20	1	19
	CPU time average (sec.)	----	230.0	224.7	118.2	4104.0	2190.7
	Generations average	----	609	487	252	4948	2405
CX + Arb20%JC	Tests with optimal solution	0	4	17	20	3	20
	CPU time average (sec.)	----	165.3	107.8	51.0	2594.0	736.4
	Generations average	----	551	292	136	3912	1011

6 Conclusion

In this paper we developed an FMC and a software tool called HybFlexGA to study the scheduling of jobs in FMC with GA. We propose a new concept of genetic operators for scheduling of jobs in FMC. With the software tool HybFlexGA, we

examine the performance of various crossover and mutation operators by computing simulations on job scheduling problems. The HybFlexGA obtained good computational results in the instances of SMTWT problems with 40, 50 and 100 jobs (see Table 3). As we demonstrated, the HybFlexGA is very efficient with the TPC4C+Arb20%JC combination. With this combination, the HybFlexGA always found more optimal solutions than with the other combinations: TPC3CV2+Arb20%JC, TPC2CV2+Arb20%JC, and so on. When we used this combination (TPC4C+Arb20%JC) in the HybFlexGA, the genetic algorithm required fewer generations and less CPU time to find the optimal solutions. The results obtained in the scheduling problems of the FMC show how efficient HybFlexGA is in solving these problems.

7 References

1. Morton, E. Thomas and David W. Pentico: Heuristic Scheduling Systems. John Wiley & Sons, (1993)
2. Potts, C.N. and L.N. Van Wassenhove: Single Machine Tardiness Sequencing Heuristics. IIE Transactions, vol. 23, no 4 (1991) pp. 346-354
3. Peter A. Huegler and Francis J. Vasko: A performance comparison of heuristics for the total weighted tardiness problem. Computers & Industrial Engineering, vol. 32, no 4 (1997) pp. 753–767
4. D.E. Goldberg: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989)
5. Ferrolho A. and Crisóstomo M.: Scheduling and Control of Flexible Manufacturing Cells Using Genetic Algorithms. WSEAS Transactions on Computers, vol. 4 (2005) pp. 502–510
6. Ferrolho A. and Crisóstomo M.: Flexible Manufacturing Cell: Development, Coordination, Integration and Control. Proc. of the IEEE 5th Int. Conference on Control and Automation, Hungary (2005) pp. 1050-1055
7. J. Oliver, D. Smith and J. Holland: A study of permutation crossover operators on the traveling salesman problem. Proc. of the Second ICGA (1987) pp. 224–230
8. Syswerda: Scheduling optimization using genetic algorithms. L. Davis (Ed.) Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York (1991) pp. 332–349
9. T. Murata and H. Ishibuchi: Positive and Negative Combination Effects of Crossover and Mutation Operators in Sequencing Problems. Proc. of the IEEE Int. Conf. Evol. Computation, Piscataway (1996) pp. 170–175
10. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html