Genetic Algorithm for Open Shop Scheduling Problem

Yacine Benziani, Imed Kacem, Pierre Laroche

Genetic Algorithm For Open Shop Scheduling Problem

Yacine Benziani[1], Imed Kacem[2], Pierre Laroche[3]

Abstract— **In this paper, we present a genetic algorithm for the open shop scheduling problem. We use a simple and efficient chromosome representation based on the job's occurrence and the fitness function reflect the length of the schedule. The solutions obtained after performing the different operators of the genetic algorithm are always feasible. Heuristic approaches are also developed to generate the initial population and to improve the obtained solutions. The algorithm was implemented and computational results show interesting result.**

I. INTRODUCTION

The open shop scheduling problem consists in sequencing a set J of n jobs on a set M of m machines to minimize the length of the schedule (the makespan). A job is a set of m operations to be processed on the associated machines without interruption in an arbitrary order (the order of these operations is a decision variable for every job).

The open shop is NP-hard. In (Gonzalez and Sahni 1976) the open shop on three machines O3||Cmax is shown to be binary NP-hard. However, O2||Cmax, the open shop on two machines, is polynomially solvable in O (n). The open shop can be formulated by a disjunctive graph. Contrarily to the job shop where the order of operations is fixed, in the open shop all arcs of such a graph are disjunctive, and we have to select the direction of all arcs of the graph. This case implies that the open shop has more decision variables and larger solution space. A complete acyclic selection represents a feasible schedule and its length is the value of the longest path in the graph (Roy and Sussmann 1964).

Only few exact methods are proposed for the general open shop problem, a branch and bound procedure using the disjunctive graph formulation is developed by Brucker et al 1997. Their algorithm is based on the branching scheme of Brucker et al (Brucker et al 1994) and the immediate selection of Carlier and Pinson (Carlier and Pinson 1989) both applied to the job shop problem. This algorithm was improved by using intelligent backtracking (Guéret et al 1998) and according to constraint propagation to reduce the search space (Dorndorf et al 2001).

Due to the limitations of exact methods in term of computation time, various heuristics and meta-heuristics were proposed to the open shop problem and many of them are an adaptation from the job shop problem. Guéret and Prins (Guéret and Prins 1998) have examined the list scheduling heuristics and proposed two new heuristics based on list scheduling with priorities and matching construction algorithm. The shifting bottleneck procedure was generalized to the open shop (Ramudhin and Marier 1996), initially developed for the job shop problem. Two constructive heuristics were presented (Bräsel et al 1993): matching algorithm based on the generation of Rank-Minimal schedules and insertion algorithm initially developed for the job shop (Werner and Winkler 1995). In (Liaw 1998) and (Liaw 1999) an iterative approach was proposed based on Benders decomposition and a tabu search. A tabu search is also considered by Taillard (Taillard 1993). A genetic algorithm was proposed to the open shop (Liaw 2000) and (Prins 2000). Recently, an ant colony algorithm with beam search was proposed (Blum2005), a particular swarm optimization is discussed (Sha and Hsu 2008) and a bee colony algorithm is developed (Huang and Lin 2011). The particle swarm of Sha and Hsu gives better results for difficult instances.

In this paper, we describe the basic definition of genetic algorithm in Section 2. In Section 3, we propose a genetic algorithm for the open shop. In Section 4, we discuss the computational results given by the genetic algorithm. And finally we conclude the paper by giving some perspectives in Section 5.

II. BASIC DEFINITIONS

In this section, we describe the principle of the genetic algorithm proposed to solve the open-shop scheduling problem.

A. Genetic Algorithm

The genetic algorithm (GA) is a generic search strategy. It is based on the natural selection and mutation of natural evolution. GA was introduced to optimization by several researchers and became practical after the publication of the book of Holland (Holland 1975). The GA is a very simple process and it is used without knowing the characteristics or the mathematical formulation of the optimization problem.

To elaborate a genetic algorithm, we need an initial population. Each individual in the population is encoded by a chromosome, which represents, in the best case, a unique coding representation. The initial population is obtained generally by heuristics and random process. Each individual is evaluated by a fitness function; this function is related to the considered objective to be optimized

As a natural evolution, the GA is composed of some operators. The first operator is the natural selection: some individuals are chosen among the best ones, to become parents for individuals of the next generation On the selected individuals a mutation operator is applied. It consists on

[1]Yacine Benziani is a member of University of North Dakota, USA mybenziani@aero.und.edu

Imed Kacem[2] and Pierre Laroche[3] are members of the LCOMS Laboratory, University of Lorraine, F-57045 Metz, France imed.kacem@univ-lorraine.fr, pierre.laroche@univ-lorraine.fr

Chromosome	2	5	4	1	9	7	8	6	3

Fig. 1: Chromosome Representation

Parent	2	5	**4**	1	9	7	8	**6**	3
Child	2	5	**6**	1	9	7	8	**4**	3

Fig. 2: Simple Mutation Operator

the permutation of chromosome elements. Another important operator is the crossover. This operator is used to obtain new individuals by recombination of the old ones. The new individuals, called children, will be added to the population by replacing the parents or the worst individuals in the population and if we want to keep a fixed number of individuals. The new children can also be added to the population without deleting the parents and the number of individuals grows for each generation. The process is repeated until the stopping criterion is satisfied. The stopping criterion can be either the number of iterations, a time criterion or the gap between a certain lower bound and the GA solution.

B. Genetic Algorithm for Scheduling Problem

The genetic algorithm was applied to a wide range of scheduling problems. The chromosome represents generally the order of jobs on machines. The crossover operator is more complicated to implement due to the infeasibility of solutions resulted from GA operators since the job apparition in the sequence could be repeated. We need to rearrange the chromosome to obtain a feasible solution.

The following example describes a chromosome of the single machine problem and different operators of the genetic algorithm, such as the mutation and crossover.

As seen in Figure 1, a chromosome is given by an order of jobs in the machine.

To perform the mutation operator, in our algorithm, we choose randomly two points in the chromosome and we swap the two associated jobs. Another way to get a mutated child is to choose randomly three points in the sequence and swap the associated jobs in circular permutation. Figures 2 and 3 illustrate these mutations.

Given two parents, the crossover operator is applied as follows: we choose randomly a point in the chromosome and we copy the first part of the first parent in the first child. We complete the remaining positions in the sequence with the missing jobs following the apparition order in the second parent. This ensures that the obtained children represent feasible solutions. We apply the same process for the second child. Figure 4 illustrates the process.

Parent	2	5	**4**	1	9	**7**	8	**6**	3
Child	2	5	**6**	1	9	**4**	8	**7**	3

Fig. 3: Multi-points Mutation Operator

Parent 1	2	5	4	1	9	7	8	6	3
Parent 2	1	7	6	9	2	3	5	8	4

Child 1	2	5	4	1	7	6	9	3	8
Child 2	1	7	6	9	2	5	4	8	3

Fig. 4: Crossover Operator

Machines / Jobs	1	2	3
1	4	5	8
2	2	9	9
3	4	7	2

TABLE I: Processing time

III. GENETIC ALGORITHM FOR THE OPEN SHOP PROBLEM

A population is an array containing several individuals. Each individual represents a solution of the open shop problem. The individuals are evaluated by a fitness function according to the objective function. At each step of the algorithm, some individuals are selected to reproduce new members by the crossover and mutation operators.

A. Representation

A solution of the open shop is defined by ordering the operations on each machine and by fixing an order of operations. In the disjunctive graph representation, a feasible schedule is a complete acyclic selection obtained by an orientation of the disjunctive arc. The makespan equals to the longest path in the oriented graph. A feasible schedule is also an assignment of starting time for all operations. A solution can be represented by different ways. An example of 3×3 machines-jobs with processing times given in Table I will illustrate these different ways:

1) Starting Time Representation: The first way is obtained by taking $n \times m$ integer value array which represents the starting times (y_i) of operations. The operations are ordered job by job; we put in the first m position the operations of the first job and so on (see Figure 5).

2) Jobs Sequence Representation: The second representation is defined by the sequence of jobs in the machine and operations in the job, called the job sequence. It consists of two 2-dimensional arrays: the first array contains the job sequence on each machine, and the second defines the precedence order of operations (see Figure 6).

3) Working Sequence Representation: In the last representation, the solution is represented by the working sequence with two $n \times m$ arrays. The first array contains the job

i	1	2	3	4	5	6	7	8	9
y_i	0	18	25	25	9	18	9	0	18
M	1	2	3	1	2	3	1	2	3
Job	1			2			3		

Fig. 5: Starting time representation

	Job Sequence				Precedence Order		
	J1	J2	J3		J1	J2	J3
M1	1	3	2	M1	1	2	3
M2	2	1	3	M2	2	3	1
M3	3	2	1	M3	3	1	2

Fig. 6: Jobs Sequence Representation

Op	1-3	1-1	1-2	2-3	2-2	3-2	2-1	3-1	3-3
Inter	3	1	2	3	2	2	1	1	3
Intra	1	2	3	2	3	1	3	1	2

Fig. 7: Working Sequence Representation

apparition called Inter-Job, and the second array contains the precedence order called Intra-Job. The Intra-Job defines the order of operations in the job, and the Inter-Job defines the order of jobs through the machines.

Following the job order in Intra-Job, the job occurrence in Inter-Job defines the apparition order of its operations and the sequence in the machines is constructed. Each job appears m times (number of job operations) (see Figure 7).

From each representation, we can find the others by constructing the schedule using the Gant Chart. The working sequence is obtained by ordering the starting time in non-decreasing order and we put the operations in the Inter-Job following this order and respecting the Intra-Job giving by job operations order.

B. Chromosome Representation and Fitness Function

In our genetic algorithm we use the working sequence representation. The chromosome is represented by two integer arrays. The first array is the Inter-Job and contains the job number, and the second is the Intra-Job and contains the machine number. Each solution has a unique coding representation and each chromosome corresponds to one solution.

The fitness function is the value of the makespan; the length of the schedule. Given a solution represented by its chromosome, we process as follows to compute the length of the schedule: we read the first operation and we set the associated job length and the loading machine with its processing time. We read the remaining operations, and at each position we update the associated job length and loading machine with the possible starting time added to the related processing time. When all operations are completed, the length of the schedule equal to the maximum of loading machine and job length.

Inter-Job	3	1	2	1	2	2	1	3	3
Intra-Job	3	2	1	2	3	1	3	2	1

Fig. 8: Chromosome

Parent

Inter-Job	3	1	2	3	2	2	1	1	3
Intra-Job	1	2	3	2	3	1	3	1	2

Child

Inter-Job	3	1	2	1	2	2	1	3	3
Intra-Job	3	2	1	2	1	3	3	2	1

Fig. 9: Mutation

C. Initial Population and Heuristics for the Open Shop

The initial population contains several different solutions. The solutions are obtained by some heuristics used to give a good gene. Some of them are obtained by using priority rules (Giffler and Thompson, 1960). The population is completed randomly to diversify the chromosomes.

In this sub-section, we will describe some heuristics for the open shop. It is very useful to improve the genetic algorithm solutions.

1) Greedy Search Technique.: This heuristic is based on the search tree generated by a branch and bound method (we do not describe the branch and bound algorithm in this paper, we refer the reader to the literature for more details (Brucker et al 1994) and (Dorndorf et al 2001)). We apply a Depth-First-Search for only $2m(n(n-1))$ (number of combinations to obtain a complete schedule) and we choose the vertex in the tree search having the best evaluation to find the first solution.

2) Neighborhood Search.: Given an initial solution for the open shop, we explore its neighborhoods by swapping two operations. After trying to swap all operations two per two, then we swap two machines and we reiterate the operations process until testing all combinations of machines.

D. Mutation Operator

The mutation operation is used to diversify the population, so that, if we are blocked in a local optima we can jump to another area in solutions domain. In the apparition order, we choose randomly two positions in the working sequence, containing two different jobs, and then we swap the selected operations. And in job precedence order, on each job we swap two machines (see Figure 9).

E. Crossover Operator

After choosing two arbitrary individuals from the population, we apply the crossover operator. We choose an arbitrary position in the apparition order of the first parent and we build the new individual by taking the first part of the first parent and we complete the apparition order by the genes appearing in the second parent. In the precedence order, we choose randomly a job; the first child obtains the first j precedence order of the first parent and we complete the precedence order by the last part of the second parent. We apply the same technique for the two parents to obtain two new individuals. In the example in Figure 10, we generate two different solutions.

Parents

Inter-Job	3	1	2	3	2	2	1	1	3
Intra-Job	1	2	3	2	3	1	3	1	2

Inter-Job	3	1	2	3	2	2	1	1	3
Intra-Job	3	1	2	1	3	2	3	2	1

Children

Inter-Job	3	1	2	1	3	3	2	2	1
Intra-Job	1	3	2	2	1	3	3	2	1

Inter-Job	3	1	2	2	3	2	2	1	3
Intra-Job	3	1	2	1	2	3	3	1	2

Fig. 10: Crossover

F. Process of the Genetic Algorithm

The genetic algorithm runs as follows: First, we produce individuals using the heuristics and we complete the population with random solutions. In our implementation we use 500 individuals. For each generation, we choose arbitrary, from the 30% best solutions, two individuals who represent the parents, on these parents, we perform the mutation and the crossover operators with a fixed probability, 0.1 for the mutation and 0.8 for the crossover, to obtain two new children. Then, we select randomly two other individuals among the 70% less well and we replace them with the new children. After that we select randomly with a probability of 10% an individual and apply the neighborhood search. We reiterate the process until the stopped condition is satisfied, in our test we stop after 1500 iterations.

IV. COMPUTATIONAL RESULTS

In this section, we tested the genetic algorithm and the heuristics on the instances of Taillard (Taillard 1993) denoted Tai *-* (*-* : number of machines and instance number), and instances of (Guéret and Prins 1999) GP *-*. The PC used is an Intel Core 2 Quad Processor CPU 2.83 GHz under Windows 7 operating system, the program is coded in Visual C++. We report in the following table results the name of the instances, the lower bound, the best known solution BKS, the GA solution and the gap computed by (C_{max} - AG)/AG.

Taillards instances are composed of four types such that the number of machines is equal to the number of jobs $n = m = 4, 5, 7, 10$, each part contains ten different instances. The lb in Table II represents the trivial lower bound; the maximum of maximum loading machines and maximum job length. The best known solution is the optimal solution and it is proven for all instances. The gap is calculated if the optimal solution is not reached. The optimal solution is obtained in all instances with 4, 5, 7 and 10 machines-jobs. We report in the table below instances with 10×10 machine-jobs.

Guéret and Prins instances are composed of four types. The number of machines is equal to the number of jobs $n = m = 4, 5, 6, 7, 8$. The value lb is the best lower bound. The optimal solution is obtained in all instances of 3×3, 4×4, 5×5, 6×6, and 7×7 machines-jobs except for one instance. We reported in tables III and IV the solutions obtained by the GA for the instances of 7×7 and 8×8.

TABLE II: Taillards instances 10*10

Instances	lb	BKS	AG
Tai 10-0	637	637	637
Tai 10-1	588	588	588
Tai 10-2	598	598	598
Tai 10-3	577	577	577
Tai 10-4	640	640	640
Tai 10-5	538	538	538
Tai 10-6	616	616	616
Tai 10-7	595	595	595
Tai 10-8	595	595	595
Tai 10-9	596	596	596

TABLE III: Guéret and Prins instances 7×7

Instances	lb	BKS	AG
GP07-01	1000	1159	1159
GP07-02	1000	1185	1185
GP07-03	1000	1237	1237
GP07-04	1000	1167	1167
GP07-05	1000	1157	1157
GP07-06	1000	1193	1193
GP07-07	1000	1185	1185
GP07-08	1000	1180	1181
GP07-09	1000	1220	1220
GP07-10	1000	1270	1270

We note that instances of Taillard are less difficult to solve than instances of Guéret and Prins. This is due to the fact that the optimal solution has the same value as the lower bound.

V. CONCLUSION

In this paper, we have presented a genetic algorithm for the open shop. The chromosome representation was helpful for implementing of the GA algorithm. It also allows the operators of GA to be simpler and avoids the occurrence of infeasible solutions. The GA gave us good computational results and it can solve many instances. The gap obtained is about 5% for the hardest instances. We have also found the optimal solution for many instances.

For future research, we will focus on solving harder instances and the implementation of parallel GA algorithm for shop scheduling problem. It is also interesting to gener-

TABLE IV: Guéret and Prins instances 8*8

Instances	lb	BKS	AG
GP08-01	1000	1130	1160
GP08-02	1000	1135	1136
GP08-03	1000	1110	1111
GP08-04	1000	1153	1153
GP08-05	1000	1218	1218
GP08-06	1000	1115	1115
GP08-07	1000	1126	1126
GP08-08	1000	1148	1148
GP08-09	1000	1114	1114
GP08-10	1000	1161	1161

alize the GA for the flexible scheduling problem since this problem has a similar formulation.

REFERENCES

[1] C. Blum *Beam-ACOHybridizing ant colony optimization with beam search: An application to open shop scheduling*. Computers & Operations Research, 32(6), 1565-1591, 2005.

[2] H. Bräsel, T. Tautenhahn and F. Werner *Constructive heuristic algorithms for the open shop problem*. Computing, 51(2), 95-110, 1993.

[3] P. Brucker, J. Hurink, B. Jurisch and B. Wöstmann *A branch & bound algorithm for the open-shop problem*. Discrete Applied Mathematics, 76(1), 43-59, 1997.

[4] P. Brucker, B. Jurisch and B. Sievers *A branch and bound algorithm for the job-shop scheduling problem*. Discrete applied mathematics, 49(1), 107-127, 1994.

[5] J. Carlier and É. Pinson *An algorithm for solving the job-shop problem*. Management science, 35(2), 164-176, 1989.

[6] U. Dorndorf, E. Pesch and T. PhanHuy *Solving the open shop scheduling problem*. Journal of Scheduling, 4(3), 157-174, 2001.

[7] B. Giffler and G.L. Thompson *Algorithms for solving production-scheduling problems*. Operations research, 8(4), 487-503, 1960.

[8] T. Gonzalez and S. Sahni *Open shop scheduling to minimize finish time*. Journal of the ACM (JACM), 23(4), 665-679, 1976.

[9] C. Guéret and C. Prins *Classical and new heuristics for the open-shop problem: A computational evaluation*. European Journal of Operational Research, 107(2), 306-314, 1998.

[10] C. Guéret and C. Prins *A new lower bound for the openshop problem*. Annals of Operations Research, 92, 165-183, 1999.

[11] C. Guéret, N. Jussien and C. Prins *Using intelligent backtracking to improve branch-and-bound methods: An application to open-shop problems*. European Journal of Operational Research, 127(2), 344-354, 2000.

[12] J. H. Holland *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.

[13] Y. M. Huang and J.C. Lin *A new bee colony optimization algorithm with idle-time-based filtering scheme for open shop-scheduling problems*. Expert Systems with Applications, 38(5), 5438-5447, 2011.

[14] C. F. Liaw *An iterative improvement approach for the non-preemptive open shop scheduling problem*. European Journal of Operational Research, 111(3), 509-517, 1998.

[15] C. F. Liaw *A tabu search algorithm for the open shop scheduling problem*. Computers & Operations Research, 26(2), 109-126, 1999.

[16] C. F. Liaw *A hybrid genetic algorithm for the open shop scheduling problem*. European Journal of Operational Research, 124(1), 28-42, 2000.

[17] C. Prins *Competitive genetic algorithms for the open-shop scheduling problem*. Mathematical methods of operations research, 52(3), 389-411, 2000.

[18] A. Ramudhin and P. Marier *The generalized shifting bottleneck procedure*. European Journal of Operational Research, 93(1), 34-48, 1996

[19] B. Roy and B. Sussmann *Les problemes dordonnancement avec contraintes disjonctives*. Note ds, 9, 1964.

[20] D. Y. Sha and C. Y. Hsu*A new particle swarm optimization for the open shop scheduling problem*. Computers & Operations Research, 35(10), 3243-3261, 2008.

[21] E. Taillard *Benchmarks for basic scheduling problems*. European journal of operational research, 64(2), 278-285, 1993.

[22] F. Werner and A. Winkler. *Insertion techniques for the heuristic solution of the job shop problem*. Discrete Applied Mathematics, 58(2), 191-211, 1995.