ICGA '91

Conventional Genetic Algorithm for Job Shop Problems

Ryohei Nakano
Takeshi Yamada
Comm. and Info. Proc. Labs, NTT
1-2356 Take, Yokosuka, 238-03, Japan

Abstract

The job shop problem (JSP) is NP-hard, much harder than the traveling salesman problem. This paper shows how a conventional Genetic Algorithm (GA) can efficiently solve the JSP. We introduce unique ideas in representation, evaluation, and survival. A solution is succinctly represented as a binary genotype even though the JSP is an ordering problem. Mostly a genotype g produced by conventional crossover is illegal, i.e., represents no feasible schedule. Therefore an evaluation function first finds a legal genotype g' as similar to g as possible, and then evaluates g' to determine the fitness of g. The fitness of g is evaluated as the total elapsed time of the corresponding schedule. In survival of genotypes, we introduce a new treatment, called forcing, that replaces the genotype g with g' when g is selected as the survivor. Forcing both quickens convergence of GAs and drastically improves solution quality. A conventional GA using the three ideas is applied to three well-known JSP benchmarks. The solution quality approaches those obtained by branch and bound methods.

1 Introduction

The job shop problem (JSP) is among the hardest combinatorial problems. Not only is it NP-complete [Garey and Johnson, 1979], but it is one of the worst NP-complete class members. The flow shop problem (FSP), a restricted version of JSP, can be reduced to the traveling salesman problem (TSP) [Reddi and Ramamoorthy, 1972]; hence, the JSP is much harder than the TSP.

Research on the JSP has been the subject of much significant literature [Muth and Thompson, 1963; Balas, 1969; McMahon and Florian, 1975; Barker and McMahon, 1985; Carlier and Pinson, 1989]. The primary algorithms to solve JSP's are the branch and bound methods. The performance of existing algorithms has been evaluated through the widely known JSP benchmarks [Muth and Thompson, 1963]. The main historical progress in solution quality will be shown later together with our results.

Research on the application of Genetic Algorithms (GAs) to JSP's is relatively recent [Davis, 1985; Liepins and Hilliard, 1987; Cleveland and Smith, 1989; Whitley *et al.*, 1989]. Moreover, all investigated the FSP, although some discussed a more realistic problem than the FSP defined in the next section. Anyway the problems investigated so far are rather simple versions of the JSP.

This paper is organized as follows. Section 2 defines the JSP addressed and Section 3 describes a binary representation of a solution genotype. Section 4 presents how to evaluate a genotype g; g is mostly illegal, i.e., represents no feasible schedule. When g is illegal, an evaluation function finds a legal genotype g' as similar to g as possible, and then evaluates g' to assess the fitness of g. The fitness of g is equal to the total elapsed time of the corresponding schedule. Section 5 discusses the forcing which means the replacement of the genotype g with g', when g is a survivor. Section 6 shows the results of experiments conducted on three well-known JSP benchmarks.

```
machine1:      111     44444333333333 66666666662222222222555
machine2:22222222444446661111115555  3
machine3:333331  2222255555555544444                          6
machine4:      3333        666         4441111111           22225
machine5:                2222222222  555553333333444444446666111111
machine6:           33333333  666666666222222222255551114444444444
```

(a) Schedule (total elapsed time = 55)

machine1 :	1	4	3	6	2	5
machine2 :	2	4	6	1	5	3
machine3 :	3	1	2	5	4	6
machine4 :	3	6	4	1	2	5
machine5 :	2	5	3	4	6	1
machine6 :	3	6	2	5	1	4

(b) Symbolic representation

job1	: 3	1	2	4	6	5
job2	: 2	3	5	6	1	4
job3	: 3	4	6	1	2	5
job4	: 2	1	3	4	5	6
job5	: 3	2	5	6	1	4
job6	: 2	4	6	1	5	3

(d) Machine sequences (given)

job1 < job2 :	110100
job1 < job3 :	011000
job1 < job4 :	110010
job1 < job5 :	111100
job1 < job6 :	110000
job2 < job3 :	101000
job2 < job4 :	111100
job2 < job5 :	111111
job2 < job6 :	111000
job3 < job4 :	111001
job3 < job5 :	111100
job3 < job6 :	111101
job4 < job5 :	110100
job4 < job6 :	111010
job5 < job6 :	101000

(c) Binary representation

Figure 1: Representations of Schedule for 6 × 6 Problem

2 Job Shop Problem

The job shop problem (JSP) to be solved has N jobs to be processed on M machines and assumes the following:

- A machine can process only one job at a time.
- The processing of a job on a machine is called an *operation*.
- An operation cannot be interrupted.
- A job consists of at most M operations.
- An operation sequence within a job, called *machine sequence*, and processing times for operations are given.
- An operation sequence on a machine, called *job sequence*, is unknown. The full set of job sequences is called a symbolic representation.
- A feasible symbolic representation is called a *schedule*.

The JSP is to find a schedule which minimizes the total elapsed time.

The flow shop problem (FSP) is a restricted JSP, where machine sequences are identical for all jobs.

3 Representation

There are at least two ways of representing a solution: symbolic and binary. As is true in the case of the TSP [Goldberg and Lingle, 1985; Grefenstette *et al.*, 1985; Oliver *et al.*, 1987], symbolic representation is also more straightforward in the JSP [Liepins and Hilliard, 1987; Cleveland and Smith, 1989; Whitley *et al.*, 1989]. Conventional GAs [Holland, 1975; Goldberg, 1986], however, are less suited to symbolic representation. For example, conventional crossover or mutation cannot be applied to symbolic representation. Hence binary representation is utilized below.

We focus our attention on a job pair [j,k]. Let the machine sequences within j and k be $[o_{j1}, o_{j2}, ..., o_{jM}]$ and $[o_{k1}, o_{k2}, ..., o_{kM}]$ respectively. Consider the following function assuming operations o1 and o2 are executed on the same machine: prior(o1,o2) = 1 if operation o1 is executed prior to o2, otherwise prior(o1,o2) = 0. Then we can get a bit vector for the job pair [j,k]:

$$[prior(o_{j1}, o_{k*}), ..., prior(o_{jM}, o_{k*})].$$

Note that operations o_{ji} and o_{k*} are executed

on the same machine. For N jobs, there exist N(N-1)/2 job pairs. Hence for N jobs and M machines, total MN(N-1)/2 bits are required to represent a solution; for example, 90 bits for N=M=6, 450 bits for N=M=10, and 950 bits for N=20, M=5.

Figure 1 shows symbolic and binary representations of a schedule for the 6 × 6 (N=6 jobs, M=6 machines) problem [Muth and Thompson, 1963]. In the binary matrix B, the rows represent the 15 job pairs and the columns represent the bit vectors. For example, B(1,1)=1 means job 1 be executed prior to job 2 on machine 3, and B(1,3)=0 means job 2 be prior to job 1 on machine 2. In the symbolic matrix, the i-th row represents the job sequence to be executed on machine i. For example, the first row indicates that on machine 1, operations are to be executed in the sequence: job1 → job4 → job3 →...

Note that the above schedule in Figure 1 is an optimal schedule. The binary representation shows that bits have a tendency to continue within a machine. That is, the results confirm the heuristic that says a good schedule tends to keep the processing priority for each job pair. We can make use of the heuristic in both genotype initialization and conventional crossover. Note also that, in a binary representation, each bit has its own meaning, as seen in nature.

4 Evaluation

As stated above, a symbolic representation can be represented in a binary form. The inverse, however, does not hold. This means that the space of binary representation properly includes the space of symbolic representation. In fact, for the 6 × 6 problem, the number of elements of the former amounts to $2^{90} \simeq 10^{27}$, while that of the latter amounts to $(6!)^6 \simeq 10^{17}$. Hereafter, a binary representation is interchangeably called a genotype. Any evaluation of a genotype should take its wider space into consideration.

The idea behind our evaluation is as follows. In general, a genotype g produced either initially or by conventional crossover is illegal, i.e., represents no schedule. Therefore, our evaluation function first finds a legal genotype g' as similar to g as possible, and then evaluates g' to determine the fitness of g. The fitness of g is evaluated as the total elapsed time of the corresponding schedule.

The procedure that creates a legal genotype g' from an illegal genotype g is called the *harmonization algorithm*. The Hamming distance is used to assess the similarity of g' to g. The harmonization algorithm goes through two phases: local harmonization and global harmonization. The former creates a symbolic representation from g, removing local inconsistency within each machine. The symbolic representation may contain global inconsistencies between machines. By removing all global inconsistencies, the latter creates a schedule from the symbolic representation. The legal genotype g' represents the schedule.

							sum
job1:	*	0	0	1	1	0	2
job2:	1	*	0	0	1	1	3
job3:	1	1	*	1	1	0	4
job4:	0	1	0	*	0	0	1
job5:	0	0	0	1	*	1	2
job6:	1	0	1	1	0	*	3

(a) Original priority

							sum
job1:	*	0	0	1	1	0	2
job2:	1	*	0	0	1	1	3
job3:	1	1	*	1	1	[1]	5
job4:	0	1	0	*	0	0	1
job5:	0	0	0	1	*	1	2
job6:	1	0	[0]	1	0	*	2

(b) After selecting job 3

							sum
job1:	*	0	0	1	1	0	2
job2:	1	*	0	[1]	1	1	4
job3:	1	1	*	1	1	1	5
job4:	0	[0]	0	*	0	0	0
job5:	0	0	0	1	*	1	2
job6:	1	0	0	1	0	*	2

(c) After selecting job 2

							sum
job1:	*	0	0	1	1	[1]	3
job2:	1	*	0	1	1	1	4
job3:	1	1	*	1	1	1	5
job4:	0	0	0	*	0	0	0
job5:	0	0	0	1	*	1	2
job6:	[0]	0	0	1	0	*	1

(d) Final priority

Figure 2 : Local Harmonization Algorithm

The *local harmonization* algorithm works for each machine separately; hence, the following description addresses just one machine. The algorithm determines a job sequence, making use of an illegal genotype g. Since g states the priority by indicating which operation is to be executed prior to the other for all operation pairs on all machines, from g we can directly get a priority matrix for each machine. Figure 2 (a) shows a priority matrix for one machine directly gained from some illegal genotype. The element (2,1) = 1 indicates that job 2 is to be executed prior to job 1 on the machine. The algorithm searches for the operation having the highest priority, and finds the job 3 operation. When there is more than one such operation, one of them is selected. After selecting the operation having the highest priority, priority inconsistency is removed, as shown in (b). By repeating the above, the algorithm next selects the job 2 operation, resulting in (c). Thus repeating the process, the algorithm finally gets the consistent priority matrix, as shown in (d). It states the job sequence should be job3 → job2 → job1 → job5 → job6 → job4. On the whole, the algorithm changed three bits of the genotype.

It is rather easy to see that the local harmonization algorithm can find a valid job sequence while changing the minimum number of bits in a genotype. The algorithm goes halfway to get a legal genotype g' from an illegal genotype g.

The *global harmonization* algorithm is embedded in a simple scheduling algorithm. First, we describe the scheduling algorithm. The following notation is introduced:

$jnext(j)$: next operation to be executed within job j,
$jnext(j).machine$: machine to execute $jnext(j)$,
$mnext(m)$: next operation to be executed on machine m.

The simple scheduling algorithm inputs a symbolic representation and given conditions, i.e. machine sequences and processing time. It polls jobs checking if any job can be scheduled, and stops when no job can be scheduled. The job j can be scheduled if the following holds:

$$jnext(j) = mnext(jnext(j).machine).$$

The algorithm unconditionally schedules a job that can be scheduled. If a symbolic representation is a schedule, the algorithm always creates the schedule. Otherwise, it terminates when it meets a deadlock. Now the scheduling algorithm is modified to call the global harmonization algorithm whenever it meets a deadlock.

The global harmonization algorithm works as follows. Each job is checked to determine how far it is between jnext(j) and mnext(jnext(j).machine) in the job sequence list. The job with the minimum distance d is selected, and the job sequence is permuted to make mnext(jnext(j).machine) jnext(j). In the permutation, d operations are shifted right by one position each, resulting in d bits of change in the genotype. The global harmonization algorithm returns control to the scheduling algorithm, which in turn continues scheduling.

Thus the simple scheduling algorithm creates a schedule in cooperation with the global harmonization algorithm. It is not always guaranteed that the input (symbolic representation) yields the output (schedule) with the minimum number of shifts. We can expect, however, they are reasonably close.

On the whole, the harmonization algorithm creates a legal g' as similar to g as possible in the sense of Hamming distance. Developing optimal solutions for large problems suffers from computational explosion, and should be avoided since the evaluation will be repeated so many times. Note that usually the total number of bits of change is less than the sum of changes incurred by local harmonization and global harmonization.

5 Forcing

The framework of the conventional GA [Holland, 1975; Goldberg, 1986] is pursued. That is, conventional crossover and mutation are used. An unusual treatment called forcing, however, is introduced in this paper. Forcing means the replacement of an illegal genotype g with a quite similar legal genotype g'.

An original illegal genotype can be considered as an inherited character, while a refined legal genotype can be considered as an acquired one. Forcing can be considered as the inheritance of an acquired character, although it is not widely believed that an acquired character is inherited in nature. Since too much forcing may destroy the whole ecology and cause premature convergence, it is limited to the cases when g is selected as the survivor.

Forcing brings about at least two merits for GAs. One is to help them converge quickly. The other is to greatly improve the solution quality. The two advantages are demonstrated in the next section.

Table 1 : Main Results for Job Shop Problems

Papers	Algorithm	6 × 6 prob.	10 × 10 prob.	20 × 5 prob.
Balas1969	BAB	55	1177	1231
McMahon1975	BAB	55	972	1165
Barker1985	BAB	55	960	1303
Carlier1989	BAB	55	930	1165
Nakano1991	GA	55	965	1215

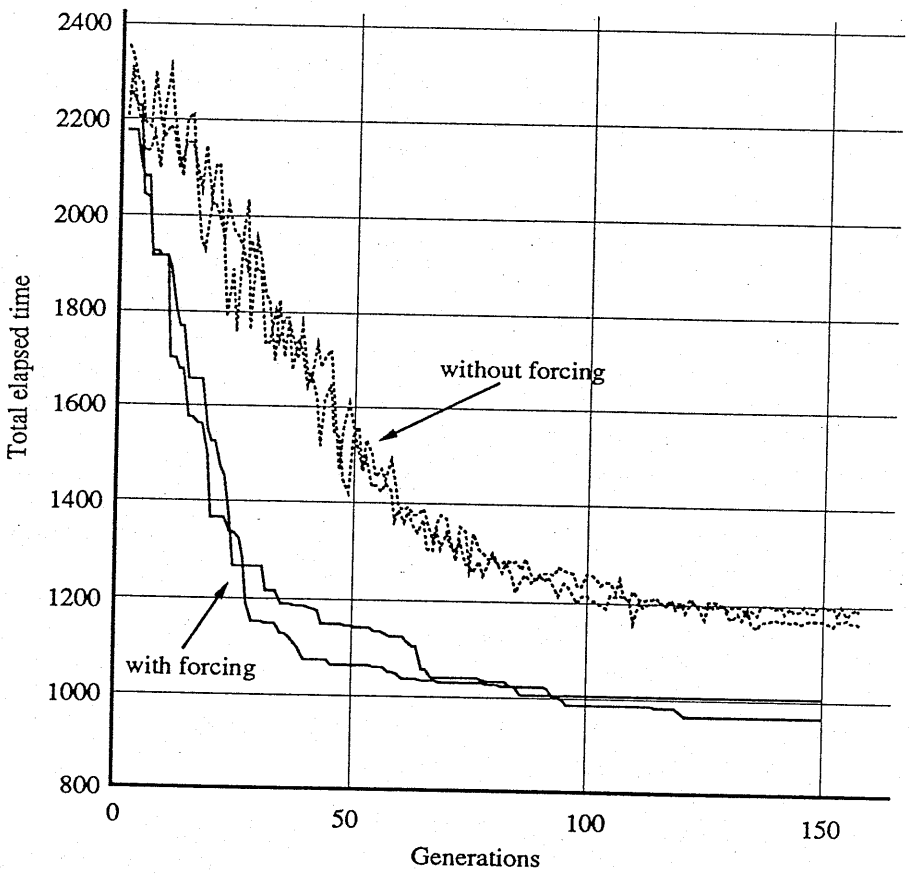

Figure 3 : Convergence of GAs for 10 × 10 problem

6 Experiments

This section shows the results of experiments conducted using three well-known JSP benchmarks [Muth and Thompson, 1963]:

- 6 × 6 (6 jobs, 6 machines) problem
- 10 × 10 (10 jobs, 10 machines) problem
- 20 × 5 (20 jobs, 5 machines) problem

Our best results (total elapsed time) are shown in Table 1 together with the historical progress in branch and bound methods.

The solution quality obtained in this paper is almost comparable with those obtained through more than 20 years of research into branch and bound methods. A population size of 1,000 is frequently used and this size is large enough to ensure a reasonably good solution for these problems.

Figure 3 shows convergence of GAs for the 10 × 10 problem. Convergence with and without forcing is displayed. The figure clearly shows that forcing both quickens convergence and drastically improves solution quality.

7 Conclusion

This paper shows how a conventional GA can effectively solve a tough combinatorial problem, the job shop problem. Three unique ideas are introduced in representation, evaluation, and survival. A binary representation pursued here makes it possible to apply conventional GAs. Even though a genotype produced by conventional crossover is usually illegal, the evaluation method presented here can evaluate it by finding a similar legal genotype. Forcing, the replacement of an illegal genotype with a legal one, improves convergence rates and solution quality. Experiments using well-known JSP benchmarks showed the solutions generated by the present approach were as good as those obtained by branch and bound methods.

Future work will include investigation of the complexity of this approach. The ideas presented in this paper can be used in symbolic representation, and such symbolic approach is also worth investigating.

Acknowledgments

The authors wish to thank Dr.Fumio Kanaya for discussion about theoretical issues, Yasuhiro Inooka for helpful information about the job shop problem, and Dr.Stephen I. Gallant for helpful comments.

References

[Balas, 1969] E. Balas. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Oper. Res.*, 17:941–957, 1969.

[Barker and McMahon, 1985] J.R. Barker and G.B. McMahon. Scheduling the general job-shop. *Manage. Sci.*, 31(5):594–598, 1985.

[Carlier and Pinson, 1989] J Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Manage. Sci.*, 35(2):164–176, 1989.

[Cleveland and Smith, 1989] G.A. Cleveland and S.F. Smith. Using genetic algorithms to schedule flow shop releases. In *Proc. 3rd Int. Conf. on Genetic Algorithms and their Applications (Arlington, Va.)*, pages 160–169, 1989.

[Davis, 1985] L. Davis. Job shop scheduling with genetic algorithms. In *Proc. 1st Int. Conf. on Genetic Algorithms and their Applications (Pittsburgh,PA)*, pages 136–140, 1985.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* Freeman and Company, New York, 1979.

[Goldberg and Lingle, 1985] D.E. Goldberg and R.Jr. Lingle. Alleles, loci, and the traveling salesman problem. In *Proc. 1st Int. Conf. on Genetic Algorithms and their Applications (Pittsburgh,PA)*, pages 154–159, 1985.

[Goldberg, 1986] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading,Mass., 1986.

[Grefenstette *et al.*, 1985] J. Grefenstette, R. Gopal, B. Rosmaita, and D.V. Gucht. Genetic algorithms for the traveling salesman problem. In *Proc. 1st Int. Conf. on Genetic Algorithms and their Applications (Pittsburgh,PA)*, pages 160–168, 1985.

[Holland, 1975] J.H. Holland. *Adaptation in Natural and Artificial Systems.* Unuv. of Michigan Press, 1975.

[Liepins and Hilliard, 1987] G.E. Liepins and M.R. Hilliard. Greedy genetics. In *Proc. 2nd Int. Conf. on Genetic Algorithms and their Applications (Cambridge,MA)*, pages 90–99, 1987.

[McMahon and Florian, 1975] G. McMahon and M. Florian. On scheduling with ready times and due dates to minimize maximum lateness. *Oper. Res.*, 23(3):475–482, 1975.

[Muth and Thompson, 1963] J.F. Muth and G.L. Thompson. *Industrial Scheduling.* Prentice-Hall, Englewood Cliffs, N.J., 1963.

[Oliver *et al.*, 1987] I.M. Oliver, D.J. Smith, and J.R.C. Holland. A study of permutation crossover operators on the traveling salesman problem. In *Proc. 2nd Int. Conf. on Genetic Algorithms and their Applications (Cambridge,MA)*, pages 224–230, 1987.

[Reddi and Ramamoorthy, 1972] S.S. Reddi and C.V. Ramamoorthy. On the flow-shop sequencing problem with no wait in process. *Operational Research Quarterly*, 23(3):323–331, 1972.

[Whitley *et al.*, 1989] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: The genetic edge recombination operator. In *Proc. 3rd Int. Conf. on Genetic Algorithms and their Applications (Arlington, Va.)*, pages 133–140, 1989.