Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Industrial Engineering 45 (2003) 597–613
computers & industrial engineering
www.elsevier.com/locate/dsw

A hybrid genetic algorithm for the job shop scheduling problems ☆

Byung Joo Park*, Hyung Rim Choi, Hyun Soo Kim

Department of MIS, Dong-A University 840, Hadan-dong, Saha-gu, Busan 604-714, South Korea

Accepted 11 July 2003

Abstract

The Job Shop Scheduling Problem (JSSP) is one of the most general and difficult of all traditional scheduling problems. The goal of this research is to develop an efficient scheduling method based on genetic algorithm to address JSSP. We design a scheduling method based on Single Genetic Algorithm (SGA) and Parallel Genetic Algorithm (PGA). In the scheduling method, the representation, which encodes the job number, is made to be always feasible, the initial population is generated through integrating representation and G&T algorithm, the new genetic operators and selection method are designed to better transmit the temporal relationships in the chromosome, and island model PGA are proposed. The scheduling methods based on genetic algorithm are tested on five standard benchmark JSSP. The results are compared with other proposed approaches. Compared to traditional genetic algorithm, the proposed approach yields significant improvement in solution quality. The superior results indicate the successful incorporation of a method to generate initial population into the genetic operators.

Keywords: Job shop scheduling problem; Genetic algorithm; Benchmark problem

1. Introduction

Scheduling problems vary widely according to specific production tasks but most are NP-hard problems. Scheduling problems are usually solved using heuristics to get optimal or near optimal solutions because problems found in practical applications cannot be solved to optimality using reasonable resources in many cases. Since the early 1980s, much interest has been devoted to the development and application of the meta-heuristic algorithms. Among the meta-heuristic algorithms,

☆ This manuscript was processed by Area Editor Subash C. Sarin.
* Corresponding author. Tel.: +82-51-200-7490; fax: +82-51-207-2827.
E-mail addresses: a967500@donga.ac.kr (B.J. Park), hrchoi@daunet.donga.ac.kr (H.R. Choi), hskim@daunet.donga.ac.kr (H.S. Kim).

0360-8352/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0360-8352(03)00077-9

the Genetic Algorithm (GA), inspired by the process of Darwinian evolution, has been recognized as a general search strategy and an optimization method which is often useful for attacking combinational problems. In contrast to other local search methods such as simulated annealinG&Tabu search, which are based on handling one feasible solution, the GA utilizes a population of solutions in its search, giving it more resistance to premature convergence on local minima. Since Davis (1985) proposed the first GA-based technique to solve scheduling problems, GA has been used with increasing frequency to address scheduling problems.

Classical GAs use a binary string to represent a potential solution to a problem. But such a representation is not naturally suited for ordering problems such as the Traveling Salesman Problem (TSP) and Job Shop Scheduling Problem (JSSP) because a classical GA representation using simple crossover or mutation on strings nearly always produces infeasible solutions. Thus, to address this problem, Bierwirth (1995) and Syswerda (1991) proposed a new chromosome representation. Dorndorf and Pesch (1993) and Yamada and Nakano (1992) used some variations on standard genetic operators and the hybridization of some existing algorithms. Bagchi, Uckun, Miyabe, and Kawamura (1991), Davis (1985) and Uckun, Bagchi, and Kawamura (1993) used a decoder or a schedule builder which performed the transition from a chromosome representation to a feasible schedule when a representation did not directly represent a schedule. Nakano and Yamada (1991) and Tamaki and Nishikawa (1992) used repair procedure to find feasible schedules from an infeasible string and a treatment called forcing. In these researches, one needs proper representation, repair procedure, forcing and decoder associated with existing algorithms to maintain a feasible chromosome, which requires additional time. If chromosome representation can always maintain feasibility after a genetic operation, the GA can be simplified because it does not need repair procedure, forcing and decoder process. Also, in most researches, the initial population is produced randomly to generate diverse individuals. However, a local search has high possibility of searching an optimal solution if a search starts from a good initial solution. Thus we can obtain a superior solution if the construction method of initial population always produces feasible, diverse and superior individuals; and genetic operators and selection method can be developed to produce a new population that is fitter than the previous one.

Also, one common problem in classical GAs is premature convergence. Although classical GAs are more resistant to premature convergence than other local search methods, GAs are not immune. One approach to reduce the premature convergence of a GA is parallelization of the GA (PGA) into disjoint subpopulations. The advantages of a PGA are preventing premature convergence and allowing speedup via the use of parallel processors. The goal of this research is to propose an efficient GA-based scheduling method addressing these concerns.

2. Genetic algorithm

As opposed to many other optimization methods, GA works with a population of solutions instead of just a single solution. GA assigns a value to each individual in the population according to a problem-specific objective function. A survival-of-the-fittest step selects individuals from the old population. A reproduction step applies operators such as crossover or mutation to these individuals to produce a new population that is fitter than the previous one. GA is an optimization method of searching based on evolutionary process. In applying GA, we have to analyze specific properties of problems and decide on

a proper representation, an objective function, a construction method of initial population, a genetic operator and a genetic parameter.

2.1. Representation

In solving JSSP by GA, first task is to represent a solution of a problem as chromosome. We utilize an operation-based representation that uses an unpartitioned permutation with m-repetitions of job numbers (Bierwirth, Mattfeld, & Kopfer, 1996). A job is a set of operations that has to be scheduled on m machines. In this formulation, each job number occurs m times in the permutation, i.e. as often as there are operations associated with this job. By scanning the permutation from left to right, the kth occurrence of a job number refers to the kth operation in the technological sequence of this job. A permutation with repetition of job numbers merely expresses the order in which the operations of jobs are scheduled.

For example, suppose a chromosome is given as [1 2 2 1 3 1 2 3 3] in 3 jobs × 3 machines problem. Each job consists of three operations, and is thereby repeated three times. Third gene of chromosome in this example is 2. Here, 2 implies second operation of job 2 because number 2 has been repeated twice. If a job number is repeated as the number of operation, the chromosome is always feasible. In the representation below, the index line refers to the kth occurrence of a job number. It is used to point to the corresponding operations in crossover.

Chromosome [1 2 2 1 3 1 2 3 3]

Index 1 1 2 2 1 3 3 2 3

We use the G&T algorithm (Giffler & Thompson, 1960) to create the initial chromosome. The schedule of G&T algorithm is produced by the process to select one operation in set G at step 4. The process is repeated each for all operations. The chromosome is created by recording job number of selected operation according to selection order.

2.2. Initial population

The initial solution plays a critical role in determining the quality of final solution in any local search. However, since the initial population has been produced randomly in most GA researches, it not only requires longer search time to obtain an optimal solution but also decreases the search possibility for an optimal solution. In this paper, we use G&T algorithm to generate initial population of good and diverse individuals. G&T algorithm may produce the easiest diverse active and non-delay schedules in JSSP. We construct initial population with these schedules (active′, active, non-delay schedule). Below is a brief outline of the G&T algorithm for obtaining active schedules.

G&T algorithm (active schedule)

Step 1.
Let C contain the first schedulable operation of each job;
Let $r_{jm} = 0$, for all operation (j, m) in C
(r_{jm} is the earliest time at which the operation (j, m) can start)

Step 2.
Compute $t(C) = \min_{(j,m)\in C}\{r_{jm} + P_{jm}\}$ ($P_{jm:}$ the processing time of operation (j,m))
And let m^* denote the machine on which the minimum is achieved
Step 3.
Let G denote the conflict set of all operations (j,m^*) on machine m^* such that $r_{jm^*} < t(C)$.
Step 4.
Randomly select one operation from G and schedule it.
Step 5.
Delete the operation from C, and include its immediate successor in C.
Step 6.
Update r_{jm} in C and return to step 2 until all operation are scheduled.

Non-delay schedules are produced by a modification of step 2 and step 3 in the above procedure of G&T algorithm.

Step 2.
Compute $t(C) = \min_{(j,m)\in C}\{r_{jm}\}$, let m^* denote the machine on which the minimum is achieved
Step 3.
Let G denote the conflict set of all operations (j,m^*) on machine m^* such that $r_{jm^*} = t(C)$.

Active′ schedules are produced by a modification of step 3 in the above procedure of G&T algorithm. The generated set of active′ schedules is a smaller set of active schedules.

Step 3.
Let G denote the conflict set of operations with the most minimal r_{jm^*} value among all operations (j,m^*) on machine m^* such that $r_{jm^*} < t(C)$.

2.3. Crossover

Since the chromosomes produced by G&T algorithm have good schedules, the genetic operator to improve temporal relationships among all operations in a chromosome is required. In this research, four modified crossovers are used for applying to PGA.

Crossover 1 arises from a modification of PPX (Bierwirth, 1996). First, the parent 1 and parent 2 are selected. Then a vector of length (j jobs × m machines) is randomly filled with elements of the set $\{1,2\}$. This vector defines the order in which genes are drawn from parent 1 and parent 2, respectively. After a gene is drawn from one parent and deleted from the other one, it is appended to the offspring chromosome. This step is repeated until both parent chromosomes are empty and the offspring contains all the genes involved. Crossover 2 arises from a modification of GOX (Bierwirth, 1995). First a substring is chosen randomly from the parent 1 of two parents. Crossover 2 implants the substring into the parent 2 at the position where the first gene of substring has occurred. Then all genes of the substring are deleted with respect to their index of occurrence in the receiving chromosome. Crossover 3 arises from a modification of GPMX (Bierwirth, 1996). First a substring is chosen randomly from the parent 1. Then all genes of the substring are deleted with respect to their index of occurrence in the parent 2. Crossover 3 implants the substring in the parent 2 at the position where it occurs in the parent 1. In

Crossover 4, a substring is chosen randomly from the parent 1. Then Crossover 4 implants the substring into the parent 2 at the position where it occurs in the parent 1. Then all genes of the substring are deleted with respect to their index of occurrence in the parent 2. Note that each crossover results in two offsprings by the same process after exchanging the parent 1 and 2 with the same interval. After two offsprings are evaluated, the better one is used as the offspring. Examples of crossover are given in Fig. 1.

2.4. Mutation

Mutation is just used to produce small perturbations on chromosomes in order to maintain the diversity of population. It is relatively easy to make some mutation operators for permutation representation. Neighborhood search-based mutation is used in this paper (Cheng, 1997). For permutation representation, the neighborhood for a given chromosome can be considered as the set of chromosomes transformable from a given chromosome by exchanging the positions of 3 genes (randomly selected and non-identical genes). Examples of mutation are given in Fig. 2.

Two types of mutation are used. One selects one chromosome after we compare neighboring chromosomes (case 2–6) except the parent chromosome. The other selects one chromosome after we compare all neighboring chromosomes. The former is used by general mutation. The latter is used after crossover and we call it 6-case mutation.

Parent 1		3 2 2 2 3 1 1 1 3
index		*1 1 2 3 2 1 2 3 3*
Parent 2		1 1 3 2 2 1 2 3 3
index		*1 2 1 1 2 3 3 2 3*
Random number		1 1 2 2 2 2 1 1 1
Crossover 1	offspring 1	3 2 1 1 2 1 2 3 3
	offspring 2	1 1 3 2 2 2 1 3 3
Crossover 2	offspring 1	3 2 2 1 2 3 1 1 3
	offspring 2	3 2 2 1 2 3 1 1 3
Crossover 3	offspring 1	3 2 2 2 3 1 1 1 3
	offspring 2	3 3 1 2 2 1 2 1 3
Crossover 4	offspring 1	3 2 3 1 1 2 2 1 3
	offspring 2	3 2 2 1 2 3 1 1 3

Fig. 1. Examples of crossover.

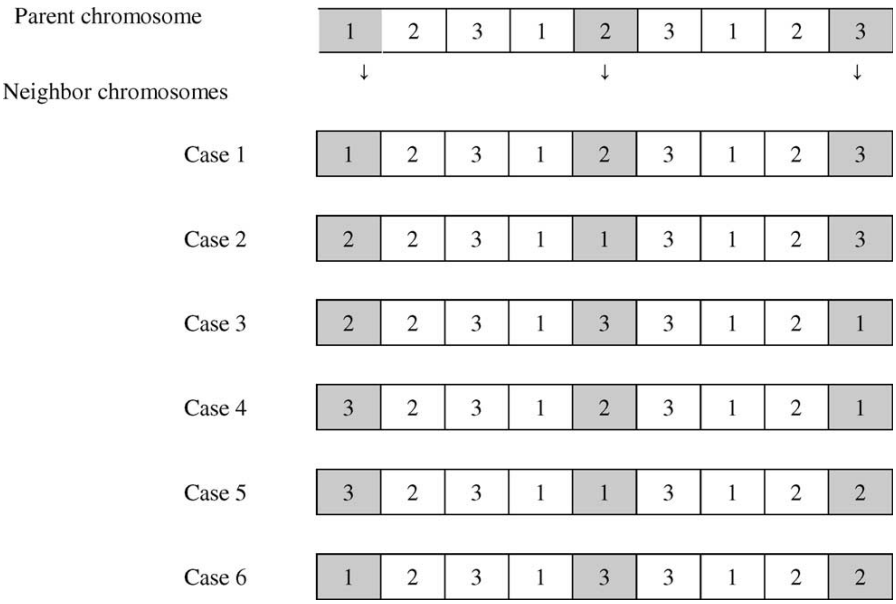

Fig. 2. An example of mutation.

2.5. Selection

In this study, we use seed selection and tournament selection. Seed selection is to introduce usual individual selection and a good individual preservation method into GA. Superior animals are mostly used as seed animals to bring forth the young at domestic animal breeding farms. Likewise, a male of parents in selection is randomly chosen in seed size involving superior individuals from ranking population when random value is smaller than a probability value (0.9), or otherwise one individual is randomly chosen from the population. A female of parents is randomly chosen in population. First we choose two individuals, then choose the fittest one if random value is smaller than a probability value (k). Otherwise, we choose the other one. Selected individuals are returned to the population and can be chosen again as a parent.

Tournament selection, which is proposed by Goldberg and Deb (1991), randomly chooses two individuals from the population, and then chooses the fittest one of two individuals if random value (r) is smaller than a probability value (k). Otherwise, the other one is chosen. Selected individuals are returned to the population and can be chosen again as a parent. Here, a probability value (k) is a parameter.

2.6. Objective functions

In JSSP, makespan represents a good performance measure. The schedule with the minimal makespan often implies a high utilization of machines. The makespan objective is selected for comparison and assessing the performance of the scheduling method in JSSP, because the objective of most JSSP is to minimize the makespan. When a chromosome is represented as a permutation type, makespan is

produced by the process that assigns operations to the machines by the technological order of each job, scanning the permutation from left to right.

2.7. Replacement

The next generation is composed through selection and genetic operator from the current generation. The next generation replaces the current generation only after the new population is completely created. We apply elitism that replaces bad individuals of next population with good individuals of current population. Some individuals are moved to next generation not through genetic operator. The replacement procedure is shown in Fig. 3.

3. Parallel genetic algorithm

In this paper, we used various initial populations, crossover and selection. It can be more efficient if these are applied to each separate subpopulation than single-population GA because this can retard premature convergence through migration among subpopulations which may be allowed to evolve independently. Island-parallel GA maintains distinct subpopulations, each of which acts as a single-population GA. At certain intervals, some individuals can migrate from one subpopulation to another.

Island-parallel GA is categorized according to a migration method, a connection scheme and subpopulation homogeneity. In migration methods, there are three types such as no migration, synchronous migration and asynchronous migration. In connection schemes, it is classified into static and dynamic connection scheme. And in subpopulation homogeneity, they are classified into homogeneous and heterogeneous subpopulation. In this paper, island-parallel GA uses synchronous migration, static connection scheme of ring type and heterogeneous subpopulations. And the population is divided into two and four subpopulations. The former is called 2-PGA, the latter is called 4-PGA. The initial population and crossover applied in PGA are shown in Table 1. The migration method is to

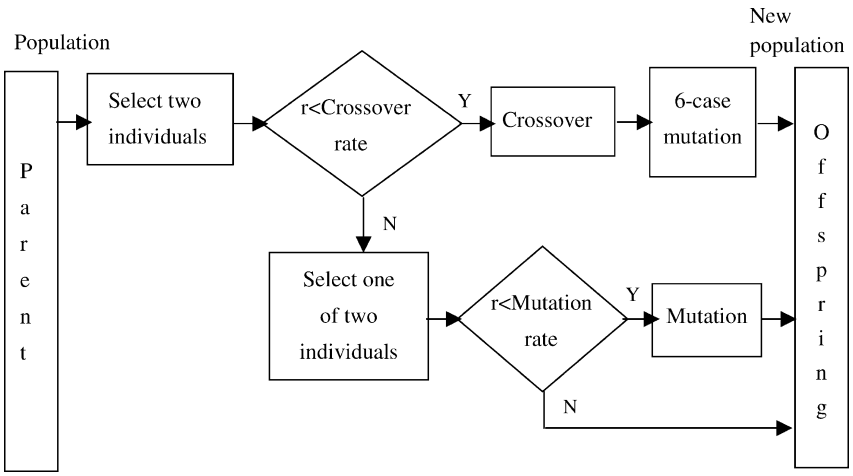

Fig. 3. The procedure of replacement.

Table 1
PGA models

Models		Crossover (selection)	Subpopulation composition	Size	Migration interval	Migration size
2-PGA	M1	1 (seed)	active′	100	50	10
		4 (tournament)	non-delay	100		
	M2	2 (tournament)	active′	100	50	10
		3 (seed)	active′	100		
4-PGA	M3	1 (seed)	active′	50	50	5
		2 (tournament)	non-delay	50		
		3 (seed)	active	50		
		4 (tournament)	random	50		
	M4	1 (seed)	active′	50	50	5
		2 (tournament)	active′	50		
		3 (seed)	active′	50		
		4 (tournament)	active′	50		

replace the individuals of low rank from connected subpopulation with the coping individuals of rank fixed from a subpopulation. And migration size and interval are decided in 10% of subpopulation and 50 generations, respectively, through the test.

4. The experiment and analysis

The performance of proposed GA-based algorithm is evaluated by comparing with other approaches in benchmark problems. In our experiments, population size is 200 and generation number is 1000. Crossover rate, selection probability in tournament selection, elitism size, and seed size are decided by experiment. The experiments are carried out on a PC with Pentium II 350 and 64MB main memory.

4.1. The parameterization of the algorithm

The parameters, such as crossover rate, selection probability, elitism size, are determined at value which produces the best value and average of results, obtained by 100 trials for a initial population on MT10 problem. In case crossover rate is 0.6, 0.7, 0.8, selection probability is 0.7, 0.75, 0.8 and elitism size is from 5 to 30 is tested. The mutation rate is applied with probability of 0.1.

From the results of Table 2, in using crossover 1, crossover rate, selection probability and elitism size is 0.7, 0.7 and 10, respectively. Crossover 2 is 0.7, 0.75, 10, crossover 3 is 0.7, 0.75, 20, crossover 4 is 0.8, 0.75, 10. In seed selection, seed size is tested from 20 to 60. From the results of Table 3, the seed size of crossover 1 and 2 is 40, crossover 3 is 60, crossover 4 is 30.

4.2. The initial population

To select creation method of the initial population in Single Genetic Algorithm (SGA), we compare results obtained, using initial population which is composed of active′, active, non-delay, random or

Table 2
The results of parameter test

Parameter		Crossover 1			Crossover 2			Crossover 3			Crossover 4		
Crossover rate	Select probability	Elitism	Average	Best solution	Elitism	Average	Best solution	Elitism	Average	Best solution	Elitism	Average	Best solution
0.6	0.75	5	966.06	937	5	961.66	937	10	966.34	937	10	960.27	951
0.7	0.75		965.17	937		961.02	951		966.84	937		958.76	945
	0.7		966.18	939		963.79	951		967.95	937		959.69	945
	0.8		965.92	937		964.54	951		967.54	937		959.26	951
0.8	0.75		967.61	939		961.32	951		966.52	937		957.49	951
0.6	0.75	10	972.16	937	10	962.87	937	15	966.52	937	20	961.27	935
0.7	0.75		966.85	937		960.84	951		965.8	937		958.83	937
	0.7		964.52	937		966.07	937		967.58	937		960.42	951
	0.8		964.67	937		963.27	951		968.68	951		960.54	951
0.8	0.75		964.59	937		961.49	951		968.91	951		960.67	951
0.6	0.75	15	971.5	937	15	962.44	937	20	966.98	951	30	962.57	940
0.7	0.75		969.27	937		963.17	951		965.5	937		962.53	951
	0.7		969.67	937		963.59	951		968.58	937		963.62	937
	0.8		972.26	942		964.35	951		967.86	951		959.52	951
0.8	0.75		969.71	936		962.39	937		966.04	937		960.05	951

Table 3
The test of seed size

Crossover	Parameter				Average	Best solution
	Elitism	Crossover rate	Selection probability	Seed size		
1	10	0.7	0.7	20	969.99	937
				30	968.54	937
				40	*962.15*	*937*
				50	964.35	937
				60	962.51	936
2	10	0.7	0.75	20	960.84	951
				30	960.57	937
				40	*960.45*	*937*
				50	962.75	951
				60	962.48	951
3	20	0.7	0.75	20	967.11	937
				30	967.8	937
				40	968.19	951
				50	969.13	951
				60	*967.79*	*937*
4	10	0.8	0.75	20	960.25	951
				30	*959.13*	*951*
				40	959.28	951
				50	959.25	951
				60	959.49	945

active′ + non-delay schedule from MT10 problem. Crossover 1, 4 and tournament selection are used in experiment. In our experiment, the parameters—elitism size (10), crossover rate (0.7), mutation rate (0.1), selection probability (0.75)—are fixed, and the initial population newly composed of a kind of schedule is used at every run.

It is 100 run times for each initial population with different schedule, the best and average of solution are compared. The initial population of active′ schedule has the best result as we can see in Table 4. And when we may solve benchmark problem by SGA, the initial population is composed of active′ schedule.

Table 4
The results generated by each initial population

Crossover		Initial population				
		Active′ schedule (200)	Non-delay schedule (200)	Active schedule (200)	Random schedule (200)	Active′ schedule (100) + Non-delay schedule (100)
1	Average	978.14	985.13	988.59	999.1	983.25
	Best Solution	936	938	939	942	937
4	Average	979.1	980.67	981.12	984.67	979.73
	Best solution	930	930	937	945	938

Table 5
The best results obtained by previous approaches on the MT problem

Authors	Problem		
	MT6 (6 × 6)	MT10 (10 × 10)	MT20 (20 × 5)
Optimum	55	930	1165
Nakano and Yamada	55	965	1215
Yamada and Nakano	55	930	1184
Gen	55	962	1175
Fang	–	949	1189
Dorndorf1 and Pesch	55	960	1249
Dorndorf2 and Pesch	55	938	1178
Croce	55	946	1178
Cheng	55	948	1196
Bierwirth	55	936	1181

4.3. Computational results of GA

We look for the best results based on 50 runs in each benchmark problem. The performance of our algorithm is compared with other proposed approaches.

4.3.1. MT problem

MT problem, which is extremely often used as benchmark problem, is proposed by Muth and Thompson (1963). MT problem has three instances: MT6, MT10, MT20. Among these, MT10 and MT20 are of particular interest because almost all JSSP algorithms proposed have used them as benchmarks. We were able to find the optimal solution for the MT6 and MT10 problems, which are 55 and 930, respectively, and the approach solution for MT20 problem, which is 1173. Table 5 summarizes the best results obtained by previous approaches for the MT problems. Table 6 shows results obtained by our algorithms.

Here, we can confirm that our algorithms achieve more improvement in solution than the previous researches. The shadow part in table represents the optimal solution or the best results.

Table 6
Computational results of the MT benchmarks

Problem	Computing time (s)	Method							
		Crossover 1		Crossover 2		Crossover 3		Crossover 4	
		Tournament	Seed	Tournament	Seed	Tournament	Seed	Tournament	Seed
MT6	38	*55*	*55*	*55*	*55*	*55*	*55*	*55*	*55*
MT10	77	936	936	*930*	937	936	936	*930*	*930*
MT20	76	1178	*1173*	1178	1178	*1173*	1178	*1173*	*1173*

Table 7
Computational results of the CAR benchmarks

Problem	Size	Optimum	Computing time (s)	Crossover 1		Crossover 2		Crossover 3		Crossover 4	
				Tournament	Seed	Tournament	Seed	Tournament	Seed	Tournament	Seed
CAR 1	11 × 5	7038	48	7038 (50)	7038 (50)	7038 (38)	7038 (40)	7038 (34)	7038 (35)	7038 (40)	7038 (43)
CAR 2	13 × 4	7166	43	7166 (33)	7166 (35)	7166 (15)	7166 (13)	7166 (11)	7166 (9)	7166 (8)	7166 (14)
CAR 3	12 × 5	7312	47	7400 (0)	7312 (1)	7399 (0)	7312 (1)	7312 (2)	7312 (1)	7312 (1)	7312 (1)
CAR 4	14 × 4	8003	44	8003 (32)	8003 (29)	8003 (24)	8003 (21)	8003 (20)	8003 (20)	8003 (25)	8003 (29)
CAR 5	10 × 6	7702	47	7732 (0)	7702 (1)	7714 (0)	7720 (0)	7720 (0)	7702 (1)	7720 (0)	7702 (1)
CAR 6	8 × 9	8313	56	8313 (10)	8313 (13)	8313 (13)	8313 (11)	8313 (15)	8313 (7)	8313 (11)	8313 (15)
CAR 7	7 × 7	6558	44	6558 (23)	6558 (19)	6558 (15)	6558 (16)	6558 (15)	6558 (21)	6558 (20)	6558 (19)
CAR 8	8 × 8	8264	52	8264 (3)	8264 (5)	8264 (4)	8264 (2)	8264 (3)	8264 (2)	8264 (3)	8264 (1)

4.3.2. CAR problem

CAR problem proposed by Carlier is a benchmark problem with various sizes (Carlier, 1988). This problem is also used to evaluate the performance of the various problem sizes. In Table 7, the numbers in the parentheses represent the frequency of optimal solutions obtained from 50 runs. From these results, we can verify that crossover 1 and seed selection are the most compatible for problems of small size. The algorithms using crossover 1 (3 or 4) and seed selection look for optimal solutions in all CAR problems.

4.3.3. ORB problem

ORB problem consists of ten 10×10 instances specially constructed to be difficult by Applegate and Cook (1991). Table 8 gives the results for this problem from our algorithms. Column 2 gives the available results from the Bottle-5 version of Adams, Balas, and Zawack (1988). Column 3 gives the results from the shuffle algorithm of Applegate and Cook. Column 4 gives the results from the tabu search of Chambers (1996). Here, we can confirm that the performance of crossover 1 is low, but everything else is superior to the other algorithms.

4.3.4. ABZ problem

ABZ problem consists of five instances which are generated by Adams et al. (1988). Adams et al. applied the shifting bottleneck (SB) algorithm to these problems. Table 9 gives the results for these problems from SBI, SBII and our algorithms.

4.3.5. YN problem

YN problem consists of four 20×20 instances which are generated by Yamada and Nakano (1992). Yamada and Nakano designed a G&T-algorithm-based operator, GA/GT crossover. They applied this operator to the four YN problems. They compared the results of GA/GT with the randomly generated 400,000 active schedules for each problem. Table 10 gives the results for these problems.

Table 8
Computational results of the ORB benchmarks

ORB problem	Bot5	Shuffle	Tabu search	Optinum	Computing time (s)	Crossover 1		Crossover 2		Crossover 3		Crossover 4	
						Torunament	Seed	Tournament	Seed	Tournament	Seed	Tournament	Seed
ORB1	1092	1070	1073	1059	78	1089	1089	1077	1072	1064	1060	1077	1060
ORB2	894	890	890	888	78	892	889	892	890	890	889	889	890
ORB3	1031	1021	1005	1005	78	1023	1025	1027	1027	1023	1022	1024	1020
ORB4	1031	1019	1013	1005	78	1014	1011	1011	1011	1019	1011	1005	1012
ORB5	896	896	899	887	78	894	894	889	889	890	894	890	889
ORB6	–	–	1026	1010	78	1023	1013	1023	1023	1023	1013	1021	1013
ORB7	–	–	397	397	78	397	397	397	397	397	397	397	397
ORB8	–	–	899	899	78	914	912	899	899	899	899	899	899
ORB9	–	–	934	934	78	947	943	934	934	934	939	943	934
ORB10	–	–	944	944	78	952	952	946	946	944	944	944	944

Table 9
Computational results of the ABZ benchmarks

Problem	Size	SB I	SBII	Optimum	Computing time (s)	Crossover 1		Crossover 2		Crossover 3		Crossover 4	
						Tournament	Seed	Tournament	Seed	Tournament	Seed	Tournament	Seed
ABZ5	10 × 10	1306	1239	1234	89	1238	1238	1236	1238	1236	1238	1236	1236
ABZ6	10 × 10	962	943	943	78	943	943	943	943	943	943	943	943
ABZ7	20 × 15	730	710	?	368	696	704	685	698	695	698	689	694
ABZ8	20 × 15	774	716	?	364	711	714	704	707	709	714	710	704
ABZ9	20 × 15	751	735	?	366	728	730	723	728	726	730	730	723

Table 10
Computational results of the YN benchmarks

Problem	Active schedule Best	GA/GT Best	Computing time (s)	Crossover 1		Crossover 2		Crossover 3		Crossover 4	
				Tournament	Seed	Tournament	Seed	Tournament	Seed	Tournament	Seed
YN1	1126	967	380	951	951	934	936	941	936	925	933
YN2	1104	945	380	956	944	956	966	966	962	966	957
YN3	1107	951	380	962	962	928	940	950	936	941	931
YN4	1202	1052	380	1069	1061	1018	1033	1039	1028	1027	1027

Table 11
The test results of PGA from MT problem

Method		Problem			
		MT10		MT20	
		Best	Average	Best	Average
2-PGA	M1	930	969.69	1173	1210.92
	M2	936	974.74	1178	1210.62
4-PGA	M3	930	964.23	1173	1207.29
	M4	936	970.12	1173	1204.44
Crossover 1	Tournament	936	977.80	1178	1245.28
	Seed	936	982.96	1173	1224.22
Crossover 2	Tournament	930	977.08	1178	1217.67
	Seed	937	975.96	1178	1217.05
Crossover 3	Tournament	936	980.91	1173	1215.85
	Seed	936	981.91	1178	1214.09
Crossover 4	Tournament	930	977.48	1173	1217.24
	Seed	930	976.92	1173	1211.85

Table 12
The test results of PGA from ORB, ABZ problem

Problem	2-PGA	4-PGA	Crossover 1		Crossover 2		Crossover 3		Crossover 4	
	M1	M3	Tourn ament	Seed	Tourn ament	Seed	Tourn ament	Seed	Tourn ament	Seed
ORB1	1061	1060	1089	1089	1077	1072	1064	1060	1077	1060
ORB2	889	889	892	889	892	890	890	889	889	890
ORB3	1025	1020	1023	1025	1027	1027	1023	1022	1024	1020
ORB4	1011	1011	1014	1011	1011	1011	1019	1011	1005	1012
ORB5	889	889	894	894	889	889	890	894	890	889
ORB6	1013	1013	1023	1013	1023	1023	1023	1013	1021	1013
ORB7	397	397	397	397	397	397	397	397	397	397
ORB8	908	899	914	912	899	899	899	899	899	899
ORB9	939	934	947	943	934	934	934	939	943	934
ORB10	944	944	952	952	946	946	944	944	944	944
ABZ5	1236	1236	1238	1238	1236	1238	1236	1238	1236	1236
ABZ6	943	943	943	943	943	943	943	943	943	943
ABZ7	694	694	696	704	685	698	695	698	689	694
ABZ8	704	704	711	714	704	707	709	714	710	704
ABZ9	726	723	728	730	723	728	726	730	730	723

Table 13
The parameters of PGA

Crossover (selection)	2-PGA					4-PGA	
	Elitism	Crossover rate	Mutation rate	Selection rate	Seed size	Elitism	Seed size
1 (seed)	5	0.7	0.1	0.7	20	2	10
2 (tournament)	5	0.7	0.1	0.75	–	2	–
3 (seed)	10	0.7	0.1	0.75	30	3	20
4 (tournament)	5	0.8	0.1	0.75	–	2	–

4.3.6. The performance evaluation of PGA

MT, ORB and ABZ problem is also used to evaluate the performance of PGA. We don't use CAR and YN problem, which is very easy and big problem. From Table 11, we can confirm that PGA improves not the best solution but the average solution. It implies that the superior solution can be obtained from fewer runs. From Table 12, we also can confirm that PGA generates generally the better solution than each SGA. The parameters of PGA are shown in Table 13.

5. Conclusion

An efficient GA-based scheduling method is developed to address JSSP in this research. In the scheduling method, the representation, which encodes the job number, is made to be always feasible.

Initial population is generated through integrating representation and G&T algorithm. The new genetic operators and selection method are designed to better transmit the temporal relationships in the chromosome. And the initial good schedules can be evolved into the better schedules. The proposed scheduling method is tested on standard benchmark JSSP. The superior results indicate the successful incorporation of generating method of initial population into the genetic operators, and show the effectiveness of the scheduling method.

The results in the five benchmark problems indicate that the case of crossover 1 and seed selection is more appropriate for small size problem like CAR problem, and the case of crossover 2 and tournament selection is more appropriate for large size problem like ABZ 7-9 and YN problem. However, the case of crossover 4 and seed selection produces the most number of optimal solutions and best solutions at all benchmark problems. Also, PGA improves not only the best solution but also the average solution from results of SGA. When time is limited PGA may produce superior solution effectively.

The superior performance of the proposed GA is obtained by the successful incorporation of the chromosome representation, the generating method of initial population, genetic operators and selection method, which are designed to better transmit the temporal relationships in the chromosome. The GA may be very useful for real world scheduling problems because of simplification based on hybridization of GA and existing algorithm and creative evolution procedure.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure in job shop scheduling. *Management Science*, *34*, 391–401.

Applegate, D., & Cook, W. (1991). A computational study of the job shop scheduling problem, ORSA. *Journal on Computing*, *3*(2), 149–156.

Bagchi, S., Uckun, S., Miyabe, Y., & Kawamura, K. (1991). *Exploring problem-specific recombination operators for job shop scheduling*. *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo: Morgan Kaufmann, pp. 10–17.

Bierwirth, C. (1995). In E. Pesch, & S. Vo (Eds.), *A generalized permutation approach to job shop scheduling with genetic algorithms, OR-Spektrum* (pp. 87–92). *Special issue: Applied Local Search, 17(213)*.

Bierwirth, C., Mattfeld, D., & Kopfer, H. (1996). In H. M. Voigt (Ed.), *On permutation representations for scheduling problems* (pp. 310–318). *Proceedings of Parallel Problem Solving from Nature IV*, Berlin, Germany: Springer.

Carlier, J., & Chretienne, P. (1988). *Problèmes d'ordonnancement, col. ERI*. Paris: Masson.

Chambers, J.B (1996). *Classical and flexible job shop scheduling by tabu search*. PhD thesis. Department of Computer Science, University of Texas.

Cheng, R (1997). *A study on genetic algorithms-based optimal scheduling techniques*. PhD thesis. Tokyo Institute of Technology.

Croce, F. D., Tadei, R., & Volta, G. (1995). A genetic algorithm for the job shop problem. *Computer and Operations Research*, *22*(1), 15–24.

Davis, L. (1985). *Job shop scheduling with genetic algorithms*. *Proceedings of the International Conference on Genetic Algorithms and their Applications*, Hillsdale: Lawrence Erlbaum, pp. 136–149.

Dorndorf, U., & Pesch, E. (1993). *Combining genetic and local search for solving the job shop scheduling problem*. *APMOD93 Proc. Preprints, Budapest, Hungary*, pp. 142–149.

Dorndorf, U., & Pesch, E. (1995). Evolution based learning in a job shop scheduling environment. *Computers and Operations Research*, *22*(1), 25–40.

Fang, H., Ross, P., & Corne, D. (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*, San Mateo: Morgan Kaufmann, pp. 375–382.

Gen, M., & Cheng, R. (1997). *Genetic algorithms and engineering design*. New York: Wiley.

Giffler, J., & Thompson, G. L. (1960). Algorithms for solving production scheduling problems. *Operations Research*, *8*, 487–503.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In G. Rawlins (Ed.), *Foundations of genetic algorithms*. Morgan Kaufmann.

Muth, J. F., & Thompson, G. L. (1963). *Industrial scheduling*. Englewood Cliffs, NJ: Prentice-Hall.

Nakano, R., & Yamada, T. (1991). *Conventional genetic algorithms for job shop problems*. *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo: Morgan Kaufmann, pp. 474–479.

Syswerda, G. (1991). Schedule optimization using genetic algorithms. In L. Davis (Ed.), *Handbook of genetic algorithms* (pp. 332–349). New York: Van Nostrand Reinhold.

Tamaki, H., & Nishikawa, Y. (1992). *A parallel genetic algorithm based on a neighborhood model and it's application to the job shop scheduling* (*vol. 2*). *Parallel problem solving from nature*, Amsterdam: North Holland, pp. 573–582.

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

Yamada, T., & Nakano, R. (1992). *A genetic algorithm applicable to large-scale job shop problems* (*vol. 2*). *Parallel problem solving from nature*, Amsterdam: North Holland, pp. 281–290.