Paper

An Improvement of Genetic Algorithms by Search Space Reductions in Solving Large-scale Flowshop Problems

Non-member Zhao Yong (Kyoto Institute of Technology)
Member Nobuo Sannomiya (Kyoto Institute of Technology)

While searching for suboptimal solutions for large-scale problems, it is critical to force search algorithms on promising regions. This paper presents genetic algorithms with search space reductions (RGAs) and their application to solving large-scale permutation flowshop problems. The reduced search spaces are defined by adding precedence constraints generated by heuristic rules. To balance between the size of reduced spaces and the risk of missing good solutions, a set of consecutively included search spaces is proposed. RGAs are implemented and their performance is tested on a large-scale flowshop problem. Primary experiments show that the RGAs outperform the standard genetic algorithms greatly. Moreover, we propose an improved uniform crossover operator which preserves the precedence constraints to focus genetic search on the specified search spaces. It is shown from computational experiments that the mechanism of search space reductions works well with GAs and RGAs outperform standard genetic algorithms significantly.

Keywords: genetic algorithm, search space reductions, permutation flowshop scheduling problem

1. Introduction

Genetic algorithms (GAs) [1] have been widely used in solving scheduling problems because of their robust performance and easy implementation. For problems which are difficult to solve by other optimization algorithms, GAs are the algorithms on hand that really work [2]. GAs keep a good balance between exploration and exploitation of the solution spaces. However, for some large-scale problems with extremely complex solution spaces, it has been suggested that the standard GAs are not efficient in getting a suboptimal solution in limited computational power [3].

Like many other evolutionary algorithms GAs start from some specified spaces and will converge finally to some good-like spaces over some generations. Sometimes these spaces may be far away from the global optimum. In this case the genetic search can not start the exploration again because of losing constantly the diversity in the population during the searching process. This is the phenomenon of premature convergence, which becomes very serious in solving large-scale problem with complex solution spaces.

Thus, tremendous research efforts have been focused on finding most promising regions from the original large solution spaces. These works can be roughly classified into two groups: the problem decomposition [4] and the search space reduction [3]. This paper aims at improving genetic algorithms with search space reduction (RGAs). The search space reductions intend to confine the original solution space to promising regions with additional constraints. For some problems hard to decompose, a heuristic-based reduction method has shown its high performance [5].

A reduced search space may not contain the optimal solution and in some bad cases the space may be far away from good solutions. If the reduced search spaces are too small, we have a risk of missing good results. On the contrary, if they are too large, the reduction mechanism does not really work. It becomes very important how the search space should be reduced. For GAs which can explore the search space, we usually choose a reasonably large reduced search spaces and let the evolutionary algorithms do the exploration. In our early paper [6], a method of choosing the size of reduced search spaces was presented. In this paper, we propose a set of consecutively included reduced search spaces which have the broad space size and good concentration on the most promising spaces. This strategy is more robust and better than the fixed reduced search spaces.

Standard genetic operators are not designed well for the reduction mechanism. That is to say, even if chromosomes of a current generation are in the reduced search spaces, those of the next generation may go far away from the spaces. To focus the genetic search on the reduced search space, special genetic operators are needed to generate the chromosomes satisfying the original constraints.

Flowshop problems viewed as sequence problems (called permutation flowshop problems) are concerned in this paper and the objective is to minimize the total tardiness. The objective of this kind of flowshop problems is based on the job order, therefore the reduced search spaces are generated by adding precedence constraints among jobs. In order to focus the genetic search on the specified reduced spaces, we test the existing crossover operators and propose an improved crossover

operator to preserve precedence constraints.

This paper is organized as follows. After a brief introduction of the permutation flowshop problems, in section 3 we generate precedence constraints by heuristic rules and reduce the original solution space by those constraints. In section 4, the design and components of RGAs are shown and the crossover operators are discussed. An improved crossover operator and an algorithm to generate the initial population are proposed. In section 5, RGAs are tested for a series of permutation flowshop problems and the results are reported. Finally, the conclusion is presented in section 6.

2. Permutation Flowshop Scheduling Problem

Permutation flowshop scheduling problems were first put forward in 1954 by Johnson and soon attracted the researcher's attention. Recently GAs have been applied to solving permutation flowshop problems and the early research has concentrated on the makespan problems. With the high demands for modernization of manufacturing system, the tardiness becomes more and more important as the objective. Reviews of applying GAs to permutation flowshop problems are found in [(2) (7)].

In an $n \times m$ flowshop problem, n jobs need to be processed on m machines in the same order. Each job i consists of m operations $o_{ij}(j = 1, ..., m)$ which must be processed on machine m_j with processing time p_{ij}. Operation $o_{ij}(j = 2, ..., m)$ can not start until $o_{i,j-1}$ is completed. Assuming that all the job sequences on the machines are the same, a permutation schedule s can be represented by a sequence $j_1^s, ..., j_n^s$, where $j_i^s \in \{1, 2, \cdots n\}$.

Each job i has the release date r_i and the due date d_i. The objective is to minimize the total tardiness C_{due} given by

$$C_{due} = \sum_{i=1}^{n} \max(c_i - d_i, 0) \quad \cdots\cdots\cdots\cdots\cdots\cdots (1)$$

where c_i is the completion time of job i. This problem is an NP-hard problem since one machine problem with the same criterion has been proved to be NP-hard [(8)].

3. Reduction of Search Space

3.1 Reduction Rules For a permutation flowshop problem with n jobs, all the $n!$ alternative sequences can be mapped to schedules and all these schedules compose the original solution space A. The size of solution space explodes exponentially with the increase of job number. But after analysis of the original data, it is found that it is unreasonable for some jobs to be processed before other jobs in a good solution. It gives us a suggestion that only some parts of solution space A is promising and the genetic search should be focused on the search spaces reduced by restricting precedence among jobs.

Defining the start time windows of jobs is a straightforward way to reduce the search space. For the first operation of job i, we predict the earliest start time $head_i$ and latest start time $tail_i$. As shown in Fig. 1, the start time window of job i is defined as the interval between $head_i$ and $tail_i$.

For the flowshop problems with release dates, the $head_i$ is set as the release date of job i and the $tail_i$ is defined as a function f of r_i, d_i and p_i, where p_i is the total processing time of all operations of job i. Thus we have

$$head_i = r_i \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (2)$$

$$tail_i = f(r_i, d_i, p_i) \quad \cdots\cdots\cdots\cdots\cdots\cdots (3)$$

$$p_i = \sum_{j=1}^{m} p_{ij} \quad \cdots\cdots\cdots\cdots\cdots\cdots (4)$$

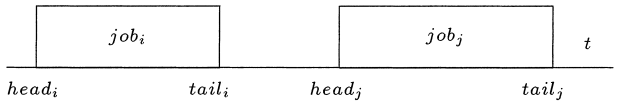


Fig. 1. Start time window

The function f enables us to adjust the size of reduced search space. In Fig. 1, the latest start time $tail_i$ of job i is smaller than the earliest start time $head_j$ of job j. It means that the job i should precede job j. Consequently if $tail_i < head_j$ then we get a precedence constraint such that job i should be processed before job j noted as $Job_i \prec Job_j$. Thus we define the following expression for three possible precedence relations between job i and job j.

$$w_{ij} = \begin{cases} 1 & \text{if } Job_i \prec Job_j \\ -1 & \text{if } Job_i \succ Job_j \\ 0 & \text{otherwise} \end{cases} \quad \cdots\cdots\cdots\cdots (5)$$

By comparing the start time windows between any pair of jobs, we get the following precedence matrix W which records all precedence constraints among jobs.

$$W = \begin{pmatrix} 0 & w_{12} & \cdots & w_{1n} \\ w_{21} & 0 & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & \cdots & \cdots & 0 \end{pmatrix} \quad \cdots\cdots\cdots\cdots (6)$$

$$w_{ij} = -w_{ji} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (7)$$

A schedule is called a legal schedule if it satisfies all precedence constraints in matrix W. The reduced search space R is composed of all the legal schedules, i.e.

$$R = \{s | s \in A,\ s \text{ satisfies constraints in (6)}\} \quad (8)$$

It is important to set the start time windows of jobs by giving the function f. If we narrow the start time windows by adjusting function f, more constraints can be got and the search space is reduced. However it runs the risk of missing good results. On the contrary, if the constraints are too loose, the search space reduction

does not really work. Rational tails can be calculated with the upper bound already known. In this paper, two parameters α and β are introduced to adjust the size of the reduced search space where $0 < \alpha < 1$ and $\beta > 0$. The tail of job i is defined as follows:

$$tail_i = r_i + \alpha(d_i - r_i - p_i + \beta p_i) \quad \cdots\cdots\cdots\cdots \quad (9)$$

A recommendation of setting these two parameters is shown in the following sections and the computation results also show that the algorithm works very well. But whether the method can effectively reduce the search space highly depends on the characteristic of job data. This method does not work well in the worst case which all jobs have the similar release or due dates. However, for the real problem in which these dates are well-distributed, the reduction method works very well.

3.2 Consecutively Included Search Space

There is no guarantee that the reduced search space contains an optimal solution. As mentioned before, choosing small search spaces may run the risk of missing good results. On the contrary, the reduction method does not work if the reduced search spaces are too large. It is hard for reduced search spaces to balance their size and the solution quality.

Therefore, in this paper a set of consecutively included search spaces is proposed. Instead of using reduced search spaces with fixed size, we define k consecutively included search spaces R_i, $i = 1, \cdots, k$ where $R_{k-1} \subset R_k$. The small search spaces intend to get good results in the early steps and the large search spaces provide opportunities to reach good solutions in the final steps.

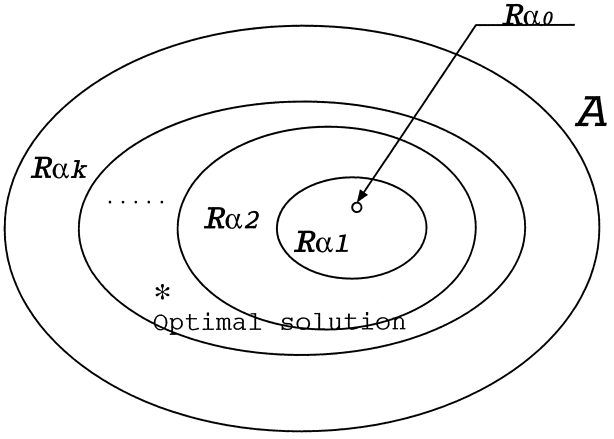


Fig. 2. Search space reduction

Fig. 2 shows the consecutively included search spaces which are generated by continuous change of the parameter α in (9). A is the original solution space which contains all the schedules. Reduced search spaces $R_{\alpha i}, i = 0, \cdots, k$ are the subset of A. If αi_1 is smaller than αi_2 then $R_{\alpha i_1}$ is a subset of $R_{\alpha i_2}$. Therefore the set of consecutively included search spaces can be represented as $S_R = \{R_{\alpha 0}, R_{\alpha 1}, \cdots, R_{\alpha k};\ \alpha 0 = 0,\ \alpha k = 1 \text{ and } \alpha 0 < \alpha 1 < \cdots < \alpha k\}$

It is noted that the reduced search space $R_{\alpha 0}$ contains only one schedule and the schedule is given by the heuristic result of ERD (earliest release date).

4. Design of GA

4.1 Main Algorithm GAs in this paper follow the frame of the standard GA (SGA) [1]. A chromosome is represented by a sequence of jobs as follows.

$$s = j_1\, j_2\, \cdots\, j_n,\ \ j_i \in \{1, 2, \cdots, n\} \quad \cdots\cdots\cdots \quad (10)$$

We use 2-tournament selection [9] as a selection operator and the insertion operator [10] as a mutation operator. The proposed algorithm is described as follows:

Main Algorithm:

Step 1. Reduce the original search space.
Step 2. $r := 0$. Generate the initial population with $POPSIZE$ chromosomes in the reduced search space.
Step 3. Select two chromosomes randomly and perform crossover operation for them. Repeat this procedure $P_c * POPSIZE$ times.
Step 4. Perform mutation operation with probability P_m for all chromosomes.
Step 5. Evaluate all chromosomes.
Step 6. Construct the new generation with $POPSIZE$ chromosomes chosen from chromosomes pool by 2-tournament selection.
Step 7. $r := r + 1$. If $r > GEN$ then stop. Otherwise go to step 3.

The proposed RGAs start from the procedure for reduced search space (either fixed search spaces or consecutively included search spaces) and the genetic search is carried out by being continuously focusing on space. For this purpose, a crossover operator is proposed as follows.

4.2 Proposition of Crossover Operators

Many crossover operators have been developed for sequence representation of chromosomes, e.g. one point crossover (1X), two point crossover (2X) and uniformed crossover (UX). For permutation flowshop problems, UX has been suggested as the best one [11]. It takes order information from one parent and inherits the remained position information from the other parent.

However, UX does not guarantee that the offsprings generated from two legal parents are still legal. Another set of crossover operators which preserve the precedence constraints were proposed in the work of applying the genetic algorithm to project scheduling problems [9]. These operators maintain the precedence constraints by preserving the common precedence. The variant forms of 1X, 2X and UX are included in that set of crossover operators. It was reported in [9] that the variant 2X and the variant UX work very well.

We consider two selected parents P1 and P2 from which two offsprings O1 and O2 will be generated. One offspring O1 is generated as follows:

Variant UX [9]:

Step1. Generate randomly a binary sequence $p_i \in \{0, 1\}, i = 1, ...n.$ $j := 1$.
Step2. If $p_j = 1$ then $j_j^{O1} := j_k^{P1}$ where k is the

lowest index such that $j_k^{P1} \notin \{j_1^{O1}, ..., j_{j-1}^{O1}\}$. Otherwise $j_j^{O1} := j_k^{P2}$ where k is the lowest index such that $j_k^{P2} \notin \{j_1^{O1}, ..., j_{j-1}^{O1}\}$.
Step3. $j := j + 1$. If $j > n$ then stop. Otherwise go to step 2.

The other offspring O2 is generated in the similar way by exchanging P1 and P2 in the above algorithm. The common precedence is transparently translated from the parents to the offsprings. It means that the offsprings of two legal parents are also legal. More details and the proofs about the precedence preserving crossover operators are found in [9].

After executing some preliminary experiments on flowshop problems [6], it is found that the variant UX sometimes performs much worse than standard UX although standard UX may generate illegal chromosomes. The reason is that the diversity among chromosomes is continuously lost in the case of applying variant UX. In this paper, a constraint maintained UX (CMUX) is proposed in such a way that the merits of both UX and precedence preservation method are gained. It starts with offsprings of standard UX and has a repair mechanism to make the offsprings legal with less possible variation. In order to accelerate this process, the sequence $(UH_1, ..., UH_n)$ is defined where $UH_i = \max(head_{j_1^o}, head_{j_2^o}, ..., head_{j_i^o})$ (o is the offspring produced with standard UX). Starting from offspring o, the CMUX works as follows :

CMUX Constraint Maintained UX:

Step 1. $UH_1 := head_{j_1^o}$ and $k := 2$.
Step 2. If $tail_{j_k^o} > UH_{k-1}$ then $UH_k := \max(head_{j_k^o}, UH_{k-1})$ and go to step 6.
Step 3. Find the maximal j such that $UH_j < tail_{j_k^o}$. If $UH_0 < tail_{j_k^o}$ then $j := 0$.
Step 4. Insert j_k^o at position $j + 1$.
Step 5. Rearrange (UH_j^o,...,UH_k^o).
Step 6. $k := k + 1$. If $k > n$ then stop. Otherwise go to step 2.

Theorem 1 The result of CMUX is a legal schedule.
Proof: After applying CMUX, the schedule o and sequence $(UH_1, ..., UH_n)$ are obtained where $UH_i = \max(head_{j_1^o}, ..., head_{j_i^o})$ and $tail_{j_k^o} > UH_{k-1}$.

Assuming there are two jobs j_a^o and job j_b^o with $a < b$ which do not satisfy the precedence constraint such that j_b^o should precede j_a^o, Then $head_{j_a^o} > tail_{j_b^o}$. Since $a < b$ we have $tail_{j_b^o} > UH_{k-1} > UH_a > head_{j_a^o}$. This is contrary to the former equation.

Fig. 3 shows a result of the Standard UX, Variant UX and CMUX for a simple permutation flowshop problem with 10 jobs($n = 10$) . In this figure, the generation of one offspring is shown. In reducing search space, we have the precedence constraint $Job_6 \prec Job_{10}$. Two parents P1 and P2 are legal solutions which satisfy that constraint. For the Standard UX, their offspring is not a legal solution. For the Variant UX, the offspring is legal. For the CMUX, we get a legal offspring with small variation from the UX result.

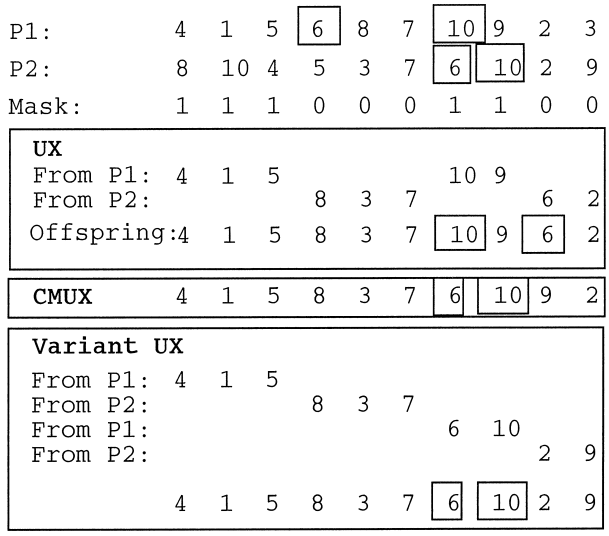


Fig. 3. Example of applying various crossover operators to a problem with constraints such as job 6 precedes job 10 and job 9

4.3 Initial Population An initial population with high quality can greatly improve the result of the genetic algorithm. In [7][12], the initial population is composed of chromosomes created by dispatch rules and randomly generated chromosomes. In this paper, step 1 and step 2 of Main Algorithm are carried out to generate the initial population from the reduced search space. Corresponding to the procedure, we have two GAs called RFGA and RCGA, where the RFGA is based on the fixed search space and the RCGA uses the consecutively included search spaces.

For both algorithms, β in (9) are manually set as constants. In RFGA, all the initial chromosomes are chosen from the same reduced search space where α is fixed as 1.

In RCGA, we generate a set of consecutively included reduced search spaces S_R. The number of search spaces in S_R is set as the population size of GAs. We choose the first chromosome from $R_{\alpha 0}$, then choose the next chromosome randomly from $R_{\alpha 1}$ and so on. The RCGA generates the initial population as follows:

RCGA:

Step 1. $i := 1$.
Step 2. $\alpha i := (i-1)/(POPSIZE - 1)$ in (9).
Step 3. Build the reduced search space $R_{\alpha i}$.
Step 4. Choose chromosome i in the reduced search space $R_{\alpha i}$.
Step 5. If $i > POPSIZE$ then stop; otherwise $i := i + 1$ and go to step 2.

4.4 Harmonization Algorithm The harmonization algorithm aims at mapping an arbitrary schedule into the reduced search space with small variations. This procedure is carried out at step 2 and the mutation operator in step 4 in Main Algorithm described in section 4.1.

The procedure is executed by repeatedly checking all precedence constraints. If the order of two jobs violates

the precedence constraints, then swap these two jobs. The process stops until all the precedence constraints are satisfied. Although this harmonization algorithm is time consuming, it is still accepted because it is executed only for generating the initial population and applying the mutation operation.

5. Computation Results

In our early paper [6], some preliminary experiments on 200×3 permutation flowshop problems with uniformly distributed due-dates demonstrated that the reduction mechanism can improve the performance of GAs significantly. In this paper, two sets of problems, a set of 200×3 problems with different data distribution and another set of larger size problems with 500×10, are generated and tested. For both sets of problems, the processing times are uniformly distributed between 0 and 100. The release dates are uniformly distributed between 0 and lower bound of makespan (LP) and the due date of job i is uniformly distributed in the range $[r_i + P_i, LP + P_i]$.

We test RCGA and RFGA by changing parameter β to clarify the robustness of RCGA and RFGA. Some experiments on crossover operators are also executed.

5.1 Parameters of GAs The settings of GAs are shown as follows: The population size of GAs is set to 500 and the algorithms stop after 5000 generations unless a specific generation is described. Crossover rate P_c is 1.0 and Mutation rate P_m is 0.05. For all GAs, the heuristic solutions ERD and EDD are added into the initial population.

Algorithms have been implemented in **C** language and run on SUN ULTRA 60 workstation. One single run of RCGA and RFGA with UX for 500×10 flowshop problem usually takes 2.9 hours, a little slower than 2.7 hours taken by standard GA. RCGA and RFGA with CMUX, which take time on crossover operator, cost 4.2 hours ($\beta = 20$)or more hours ($\beta < 20$). Considering the longer computation time, all the computation results are based on the average of 5 runs on each of 5 randomly generated problems.

5.2 RCGA, RFGA vs. SGA Fig. 4 shows the variation of suboptimal objective value with generation for RCGA, RFGA and SGA with UX for two sets of problems. The generation is extented to 20000 in solving the 500×10 problems. For all problems including problem in [6], RFGA outperforms SGA and RCGA outperforms RFGA. RCGA converges very fast and the final result is far better than RFGA and SGA. As an example, for 200×3 problems, RCGA takes 22 minutes to converge at 2000 generation and its result is almost one third of the result obtained by RFGA at 5000 generation with 55 minutes.

5.3 Reducing search spaces As mentioned in section 3, the parameter β which determines the size of reduced search spaces is difficult to be set. This experiment is carried out to investigate the effect of β on RFGA and RCGA. In this experiment, we choose the standard UX as the crossover operator.

Fig. 5 shows the variation of suboptimal objective

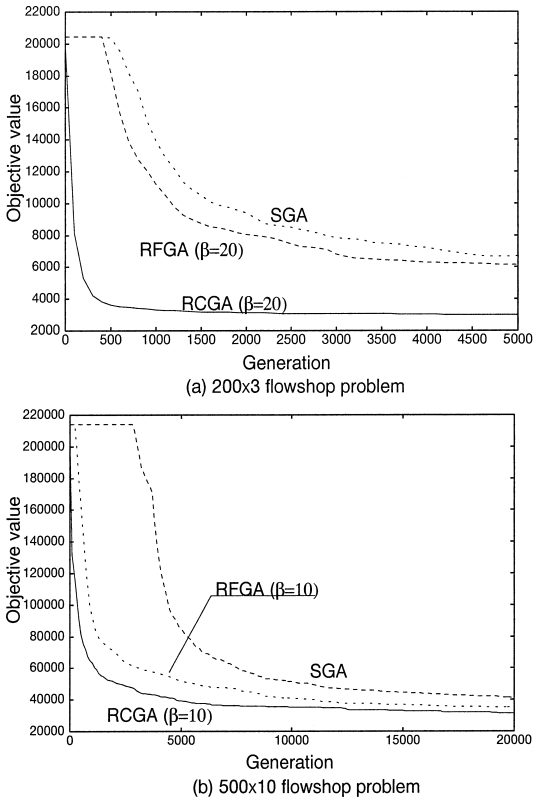


Fig. 4. Convergence curves for various GAs

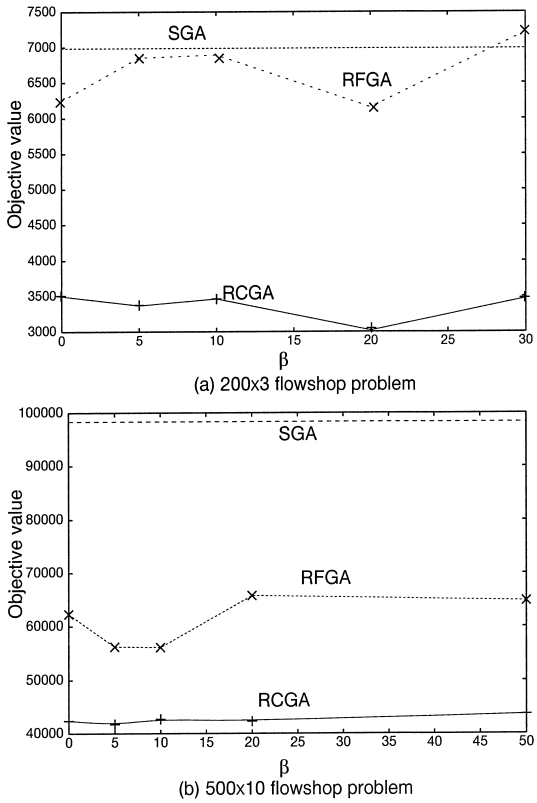


Fig. 5. Variation of suboptimal objective value with β

value with β for RCGA and RFGA. It is observed that the result of RCGA is better than those of RFGA and SGA for all β. RFGA makes only a little improvement on SGA for 200×3 problem. It is also observed that RCGA is robust for β while RFGA depends on β a little. To conclude, the solution quality for all algorithms is quite steady in wide range of β.

5.4 Comparison of Crossover Operators

Experiments are also carried out to test crossover operators. The computation results show that for all β and for both RCGA and RFGA, CMUX always outperforms standard UX and works very well in several cases although CMUX may need more computation time. Fig.6 shows the average convergence curves for various crossover operators when $\beta = 20$ for 200×3 problems.

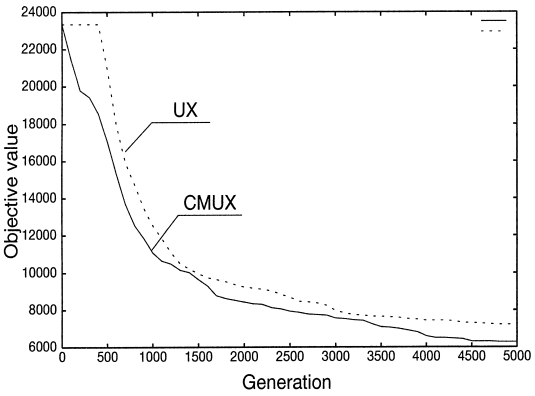


Fig. 6. Convergence curves for various crossover operators ($\beta = 20$, 200×3 problem)

5.5 Analysis The reduction mechanism performance is different with data distribution and size. For instance, the 200×3 problems in this paper seem more difficult for RFGA than 200×3 problems in [6].

Despite of the different performances of RGAs (RFGA and RCGA) on different sets of problems, they have the following two common points:

1. The reduction mechanism improves the solution quality and convergence speed.
2. RCGA is robust and good as compared with RFGA.

6. Conclusion

GAs have been improved by a technique of reducing the search space for solving large-scale flowshop scheduling problems. The reduced search spaces are defined by adding precedence constraints generated by heuristic rules. In this paper, consecutively included search spaces have been used more effectively than reduced search spaces with fixed size.

GAs with search space reduction have been tested for some sets of flowshop problems. The computation results show that the performances of GAs are improved significantly and GAs with the consecutively included search spaces are more robust and better .

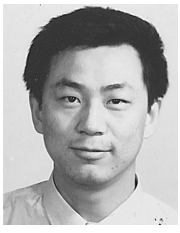
Some genetic operators maintaining genetic search on the reduced search spaces have been proposed and tested. The experiment results show that genetic operators designed carefully can be helpful in obtaining high quality solutions.

(Manuscript received October 24, 2000, revised February 15, 2001)

References

(1) D.E.Goldberg, *Genetic Algorithm in Search, Optimization, and Machine learning*, Addison-Wesley, 1989.
(2) M.Gen and R.Cheng, *Genetic Algorithm & Engineer Design*, A Wiley-Interscience Publication, 1997.
(3) S.Chen and S.F.Smith, "Improving genetic algorithm by search space reductions" , Proceedings of the Genetic and Evolutionary Computation Conference, Vol. 1 pp. 135-140, 1999.
(4) N.Sannomiya, H.Iima, K.Ashizawa and Y.Kobayashi, "Application of genetic algorithm to large-scale scheduling problem for a metal mold assembly process", Proceedings of 38th IEEE Conference on Decision and Control, pp. 2288-2293, 1999.
(5) N.Ascheuer, "Hamiltonian path problems in the on-line optimization of flexible manufacturing systems", Ph.D.Thesis, University of Technology Berlin, Germany, 1995.
(6) Y.Zhao and N.Sannomiya, "A method for solving large-scale flowshop problems by reducing search space of genetic algorithms", Proceedings of IEEE International Conference on System, Man and Cybernetics-2000, pp. 1176-1781, 2000.
(7) T.P.Bagchi, *Multiobjective Scheduling by Genetic Algorithm*, Kluwer Academic Publishers, 1999.
(8) M.R.Garey and D.S.Johnson, *Computers and Intractability*,W.H.Freeman and Company, 1996.
(9) S.Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling", Naval Research Logistic, Vol. 45 pp. 733-750, 1997.
(10) V.A. Armentano and D.P.Ronconi, "Tabu search for total tardiness minimization in flowshop scheduling problems", Computers & Operations Research, Vol. 26 pp. 219-235, 1999
(11) S.Rana, A.E. Howe, L.D. Whitley and K.Mathias , "Comparing heuristic, evolutionary and local search approaches to scheduling". Third Artificial Intelligence Planning Systems Conference, 1996.
(12) C.Reeves, "A genetic algorithm for flowshop sequencing", Computers and Operations Research, Vol. 22 pp. 5-13, 1995.

Yong Zhao (Non-member) received the M.E.in Electronics and Information Science from Xi'an Jiaotong University, China in 1997. Since 1999, he has been a doctor-course student at Division of Information and Production Science, Graduate School of Kyoto Institute of Technology. His research interests include combinatorial problems, genetic algorithms and production scheduling.



Nobuo Sannomiya (Member) received the B.E., M.E. and D.E. degrees in Electrical Engineering all from Kyoto University in 1962, 1964 and 1969, respectively. Since 1986 he has been a Professor at Department of Electronics and Information Science, Kyoto Institute of Technology. His present research interests include modeling and optimization techniques, and their applications. Especially his research works are directed toward genetic algorithm approach to the optimal scheduling of manufacturing systems.