First IEE/IEEE International Conference on Genetic ALgorithms in Engineering Systems:
Innovations and Applications (GALESIA '95), 12-14 September 1995, Sheffield, UK *(pp. 146–151)*

A Genetic Algorithm with Multi-Step Crossover for Job-Shop Scheduling Problems

Takeshi YAMADA and Ryohei NAKANO

NTT Communication Science Laboratories
2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 JAPAN
E-mail: {yamada,nakano}@cslab.kecl.ntt.jp

Abstract
Genetic Algorithms (GAs) have been designed as general purpose optimization methods. GAs can be uniquely characterized by their population-based search strategies and their operators: mutation, selection and crossover. In this paper, we propose a new crossover called *multi-step crossover* (MSX) which utilizes a neighborhood structure and a distance in the problem space. Given parents, MSX successively generates their descendents along the path connecting the both of them. MSX was applied to the job-shop scheduling problem (JSSP) as a high-level crossover to work on the critical path. Preliminary experiments using JSSP benchmarks showed the promising performance of a GA with the proposed MSX.

1. Background

The crossover operator has been considered to be the central component of genetic algorithms (GAs) and makes GAs distinctively different from other problem solvers. By using this operator a pair of solutions (parents) generates new solutions (offsprings) by mutually exchanging and recombining information. A simple GA uses a bit string as a genotype representation and a bit manipulation crossover e.g., a 1- or 2- point or a uniform crossover. The increasing complexity of problems to be solved, however, has made the simple approach difficult to use.

Now, more powerful and tailored representations and their corresponding crossover operators should be constructed to effectively solve more difficult problems. For example, permutation of the symbols denoting visiting cities might be used for TSP (traveling salesman problem), and permutation of the symbols denoting jobs or Gantt chart representations might be used for a scheduling problem[11, 16, 4].

Recently, it has become clear that the combination of a GA and local search, such as local hill-climbing or neighborhood search, works quite well. In this case, an offspring obtained by the crossover operator is not used in the next generation directly but used as a seed of the subsequent local search. The local search moves the offspring from its initial point to the nearest locally optimal point, which is used in the next generation. This approach is called as *genetic local search*[9, 14]. In a simple GA framework, the main role of the crossover is to function as a search engine; by piling up good *building-blocks*, better strings can be constructed. But here, the local search plays the leading part and the crossover helps to find new starting points for the local search.

This paper proposes a new operator called *multi-step crossover* (MSX) which is based on the idea of transitions (or moves) of neighborhood search. MSX itself can be defined in a problem independent manner and is easy to implement even if the problem to be solved is complicated. To show how MSX works, especially when combined with local search, MSX was tailored to the job-shop scheduling problem (JSSP) employing a critical path-based neighborhood. A GA with this tailored high-level MSX (GA/MSX) was evaluated by using well-known Muth and Thompson's benchmark problems[10].

2. Neighborhood Search

Neighborhood search is a widely used technique to solve combinatorial optimization problems. A solution x is represented as a point in the search space, and a set of solutions associated with x is defined as neighborhood $N(x)$. $N(x)$ is a set of feasible solutions capable of being reached from x by exactly one transition, a single perturbation of x.

Neighborhood search can be categorized by its choice criteria for selecting a new point y from $N(x)$ as the next search point; for example, a descent method selects the best point in the neighborhood, simulated annealing probabilistically selects a point and tabu search selects the best point in the neighborhood that is not on a tabu list. It is therefore very simple and easy to implement neighborhood search, although an extremely long time is taken to find the global optima. This time requirement prevents the search process from being trapped in a deep local optimum.

The outline of the neighborhood search for minimizing $V(x)$ is described in Algorithm 2.1, where x denotes a point in the search space and $V(x)$ denotes its evaluation value.

Algorithm 2.1 Neighborhood search

1. Select a starting point: $x = x_0 = x_{best}$.
2. If $V(x) < V(x_{best})$ then set $x_{best} = x$.
3. Select a point $y \in N(x)$ according to a given criterion based on the value $V(y)$. Set $x = y$.
4. Repeat 2 to 3 until some termination condition is satisfied.

The differences and similarities between the neighborhood search and the simple GA are extensively discussed by Reeves in [12]. In the next section we propose a new crossover operator for more complicated problems such as combinatorial optimizations by more aggressively adopting the idea of neighborhood search.

3. Multi-Step Crossover

Multi-step crossover (MSX) is defined by using the distance and the neighborhood structure in the search space.

The distance is used to measure the similarities between solutions and to determine the search direction in the crossover. For example, the Hamming distance can be defined in the bit string space $\{0, 1\}^n$ (n is the number of bits). In the search space of a sequencing problem, such as TSP or JSSP, a permutation-based distance can be naturally introduced. The distance in the JSSP space will be described later. By introducing the distance, the neighborhood $N(x)$ of a point x can be interpreted as a set of points close to x. It can also be said that the closer an offspring generated by the crossover is, the more it inherits its parents' characteristics.

Let the solutions of two parents be p_1 and p_2, and let $N(p)$ be the neighborhood of p. The basic idea of MSX is to evaluate a point $x \in N(p_1)$ not by the objective function $V(x)$ but by the distance $d(x, p_2)$ between x and p_2. Starting from p_1, x is modified in a step-by-step manner and uni-directionally approaches p_2. In the process, x gradually loses the characteristics of p_1 and gradually obtains those of p_2. After a certain number of iterations, the resulting new solutions should contain parts of both p_1 and p_2 although in different ratios. The outline of this procedure is described in Algorithm 3.1.

Algorithm 3.1 Uni-directional MSX

1. Set $x = p_1$. Initialize the offspring list $OL = \emptyset$.
2. Select a point $y \in N(x)$ according to a given criterion based on $d(y, p_2)$. Add y to the list OL. Set $x = y$.
3. Repeat 2 until some termination condition is satisfied.

Many variations to this algorithm can be considered. The criterion used in step 2, for instance, can vary according to the problem like in neighborhood search. In this paper, the following simple case is considered: select a point $y \in N(x)$ deterministically such that $y = \arg\min_{z \in N(x)}\{d(z, p_2)\}$. Nonetheless, stochastic method using the same criterion (which may be more appropriate from the viewpoint of the "emergent property" of the evolutionary computation) or another criterion such as the weighted sum of V and d can be considered instead. In either case, one or some of the members satisfying certain criteria are selected from OL for the next generation. In this paper, a member that is superior to or almost equally distant from both parents is selected from OL as follows:

- Select a point $y \in N(x)$ such that $y = \arg\min_{z \in N(x)}\{V(z)\}$, ***if*** $V(y) < \min\{V(p_1), V(p_2)\}$.
- Select a point $y \in N(x)$ such that $y = \arg\min_{z \in N(x)}\{|d(p_1, z) - d(z, p_2)|\}$, ***otherwise***.

The above uni-directional procedure can be converted to a bi-directional one i.e., symmetric to both p_1 and p_2 as follows:

Algorithm 3.2 Bi-directional MSX

1. Set $i = 1$. Initialize the offspring list $OL = \emptyset$.
2. Select a point $p_{i+2} \in N(p_i)$ such that $p_{i+2} = \arg\min_{z \in N(p_i)}\{d(z, p_{i+1})\}$.
   If $d(p_{i+2}, p_{i+1}) > d(p_i, p_{i+1})$ then stop; otherwise add p_{i+2} to OL.
3. Set $i = i + 1$ and go to step 2.

Figure 1 illustrates Algorithm 3.2. The algorithm stops when the two solutions p_i and p_{i+1} become so close that no more moves are possible for both parents to get closer. Like in the uni-directional case, a member that is superior to or almost equally distant from both is selected from OL for the next generation.

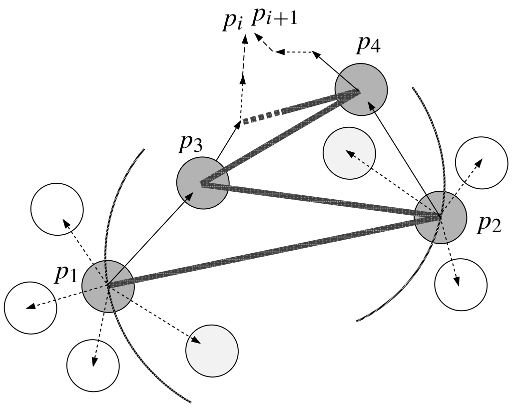

Figure 1: Bi-directional MSX

If MSX is used in combination with a local search, then because of the local optimality of the parents, it is desirable to have an offspring that is not very close to either parent. Otherwise, a subsequent local search may bring it back to the parent's position again.

Algorithm 3.3 Multi-step mutation (MSM)

1. Set $x = p$. Initialize the mutant list $ML = \emptyset$.
2. Select a point $y \in N(x)$ such that $y = \arg\max_{z \in N(x)} \{d(z, p)\}$. Add y to the list ML. Set $x = y$.
3. Repeat 2 for a fixed iteration count.

A mutation operator called *multi-step mutation* (MSM) is also defined based on the same idea as MSX. Let p be an individual to which the mutation is applied. Starting from p, an individual x is repeatedly modified and goes away from p. The outline of MSM is described in Algorithm 3.3. A member that is superior to or the most distant from p is selected from ML for the next generation in the same way as in MSX.

4. Job-Shop Scheduling Problem

The $n \times m$ *minimum-makespan* general job-shop scheduling problem can be described by a set of n jobs that is to be processed on a set of m machines. Each job has a technological sequence of machines to be processed. Each *operation* requires the exclusive use of each machine for an uninterrupted duration called the *processing time*. The time required to complete all jobs is called *makespan*. The objective when solving or optimizing this general problem is to determine the processing order of all operations on each machine that minimizes the makespan.

The JSSP is not only $\mathcal{NP}$-hard , but is extremely difficult to solve optimally. To solve JSSPs, exhaustive search algorithms based on branch and bound methods have been exhaustively studied. Recently, approximation algorithms such as simulated annealing (SA), genetic algorithms (GAs) and tabu search have also been applied with good success [1, 11, 16, 17, 13].

4.1. JSSP Solution Space and the Distance

A JSSP is often described by a disjunctive graph $G = (V, C \cup D)$, where

- V is a set of nodes representing operations of the jobs together with two special nodes, a *source* (0) and a *sink* $\star$, representing the beginning and end of the schedule, respectively.
- C is a set of conjunctive arcs representing technological sequences of the operations.
- D is a set of disjunctive arcs representing pairs of operations that must be performed on the same machines.

The processing time for each operation is the weighted value attached to the corresponding nodes.

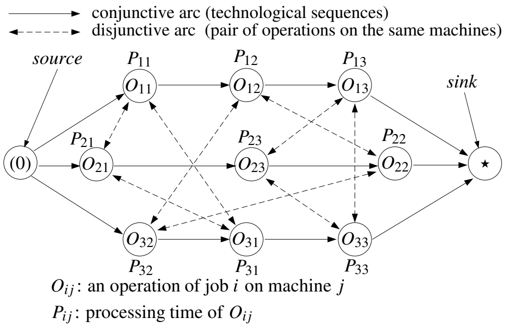

Figure 2: Disjunctive graph G of a 3×3 problem

Scheduling is to define the ordering between all operations that must be processed on the same machine, i.e., to fix precedences between these operations. In the disjunctive graph model, this is done by turning all undirected (disjunctive) arcs into directed ones. The set of all directed arcs selected from disjunctive arcs is called a *selection*.

A selection S defines a feasible schedule if and only if the resulting directed graph is acyclic. For such a case, S is called a *complete selection*. A complete selection and its corresponding feasible schedule can be used interchangeably and represented by the same symbol S.

Makespan is given by the length of the longest weighted path from source to sink in this graph. This path $\mathcal{P}$ is called the *critical path* and is composed of a sequence of *critical operations*. A sequence of consecutive critical operations on the same machine is called a *critical block*.

The distance between the two schedules S and T can be measured by the number of differences in the processing orders of operations on each machine. In other words, it can be calculated by summing the disjunctive arcs whose directions are different between S and T. We call this distance the *disjunctive graph* (DG) *distance*.

4.2. Critical Block Neighborhood

A set of solutions of a JSSP can be mapped to the space of bit strings[11]. In this case the neighborhood of a schedule S can be defined as a set of all bit strings whose DG distances from S are exactly one. Although this approach is simple and straightforward, it is not very efficient because the map is not surjective and the image is limited to some subspace of the bit string space. Some members in the neighborhood cannot be mapped to job sequences or feasible schedules. It is more efficient to limit the search to the space of all feasible schedules such as the set of all active schedules[16].

It might be still more efficient to limit the search to the critical path level. In fact, a transition operator that exchanges a pair of consecutive operations in a critical block and forms a neighborhood has been used in literature[8]. The transition operator was originally defined by Balas in his branch and bound approach[2]. In this neighborhood, the distance between S and any element in $N(S)$ is always one (so $N(S)$ is a subset of the bit string neighborhood described above). We call this the *adjacent swapping* (AS) neighborhood.

Another very powerful transition operator in the critical path level is proposed in [7, 3]. The transition operator permutes the order of operations in a critical block by moving an operation to the beginning or end of the critical block, thus forming the *CB neighborhood*. In this neighborhood, the distance between S and any element in $N(S)$ can vary depending on the position of the moving operation. It has been experimentally shown in [17] that SA using the CB neighborhood is more powerful than SA using the AS neighborhood. Thus, the CB neighborhood may as well be investigated in the GA context. Figure 3 illustrates how the two transition operators work.

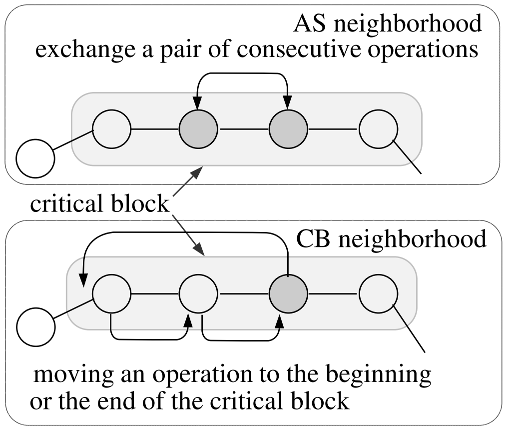

Figure 3: Permutation of operations on a critical block

4.3. GA/MSX for JSSP

The bi-directional MSX is applied to JSSP using the CB neighborhood and the DG distance previously defined. Algorithm 4.1 describes the outline of the GA/MSX routine for a JSSP using the steady state model proposed by Whitely [15]. The local search used in the algorithm is the neighborhood search described in Algorithm 2.1 with a descent method using the CB neighborhood.

Algorithm 4.1 GA/MSX for JSSP

1. Initialize population: generate a set of randomly generated schedules and apply the local search to each member in the set.
2. Select two schedules S, T from the population randomly with some bias depending on their makespan values.
3. If the DG distance between S, T is shorter than some predefined small value, apply MSM to S and generate U, then go to step 5.
4. Apply MSX to S, T using the CB neighborhood $N(S)$ and the DG distance, and generate a new schedule U.
5. Apply the local search to U and generate U'.
6. If U''s makespan is shorter than the worst in the current population, replace the worst individual with U.
7. Repeat steps 2 to 6 until no improvement is observed after a certain number of evaluations.

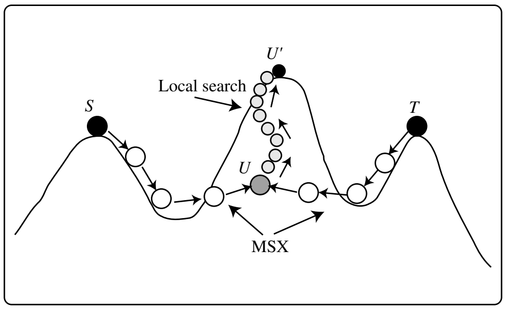

Figure 4: U' is generated from S and T by MSX followed by a local search.

Aarts *et. al* also proposed a crossover that works on the critical path in a simpler manner with limited success[1]. Figure 4 illustrates how GA/MSX searches for a new local optimum in a problem space.

5. Experimental Results

The performance of GA/MSX was tested by running several simulation trials using well-known Muth and Thompson's (MT) benchmark problems[10].

Table 1 summarizes the makespan performance of GA/MSX, a simulated annealing method using the same CB neighborhood structure (CBSA)[17], and other GA methods published so far in literature for MT10×10 and MT20×5 problems. The GA methods include GA/GT+ECO [4], PGA+SBP[5] and GVOT[6]. For GA/MSX, all runs were done on a SUN SPARC station 10 (SS10) and the programs were written in C language. As the table shows, GA/MSX outperformed the other methods in terms of the best and average performance except for the MT10×10 problem with CBSA. It should be noted, however, that the superiority of GA/MSX cannot be unconditionally claimed because the machines used, the cpu times required and the numbers of runs differed.

Figure 5 shows the average time evolution of the makespan for GA/MSX applied to the MT10×10 problem with population sizes of 100 and 500 respectively. The abscissa shows the cpu time in seconds, and the ordinate shows the best makespan of the current population averaged over ten runs with different random number seeds.

6. Discussion

The multi-step crossover (MSX) is proposed to solve combinatorial optimization problems by being used in a GA. MSX uses a neighborhood

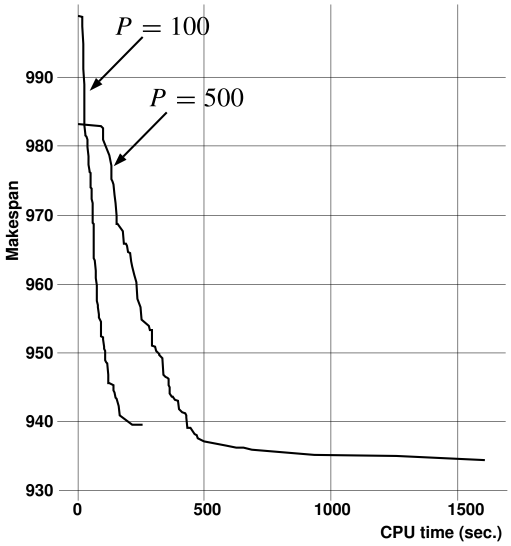

Figure 5: Average time evolution of ten runs for MT10×10 problem with population sizes (P) of 100 and 500.

structure and a distance in the problem space. Although the evaluation cost of the crossover operation increases with MSX, it carefully recombines the characteristics of both parents in a step-by-step manner and generates a new individual that is better than or almost equally distant from both of the parents. MSX is especially useful when combined with a local search, because it exploits a good new starting point for the subsequent local search.

We applied GA/MSX to the job-shop scheduling problem, one of the most difficult $\mathcal{NP}$-hard combinatorial problems known today. The preliminary experiments demonstrated that GA/MSX is comparable to or outperforms SA using the same neighborhood structure and other GA methods. However, further research including the application of GA/MSX to more difficult JSSP benchmark problems or to other kinds of combinatorial problems is necessary to show the full capabilities of MSX.

References

[1] E.H.L. Aarts, P.J.M. van Laarhoven, J.K. Lenstra, and N.L.J. Ulder. A computational study of local search algorithms for job shop scheduling. *ORSA J. on Comput.*, 6(2):118–125, 1994.

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

[3] P. Brucker, B. Jurisch, and B. Sievers. A

Table 1: Performance comparisons using MT benchmark problems

Prob	Method	Best	Avg	Var	Pop	cpu time	machine	runs
MT10×10	CBSA	930	930.8	2.4	-	44m36s	SS2	10
	GA/GT+ECO	930	963	14	2025	5m	SS2	200
($Opt = 930$)	PGA+SBP	930	947	8.2	100	2.3m	SS10	200
	GVOT	949	977	?	500	25m	SUN4	?
	GA/MSX	930	934.5	5.1	500	11m39s	SS10	10
MT20×5	CBSA	1178	1178	0	-	38m18s	SS2	5
	GA/GT+ECO	1181	1213	16	5041	30m	SS2	200
($Opt = 1165$)	PGA+SBP	1165	1188	10.3	100	2.3m	SS10	200
	GVOT	1189	1215	?	500	25m	SUN4	?
	GA/MSX	1165	1177.3	4.2	100	10m54s	SS10	10

Pop: population size, cpu time: average cpu time, machine: the machine used for the experiments, runs: number of runs, SS2(SS10): SUN SPARC Station 2(10)

branch & bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127, 1994.

[4] Y. Davidor, T. Yamada, and R. Nakano. The ecological framework ii: Improving ga performance at virtually zero cost. In *5th ICGA*, pages 171–176, 1993.

[5] H. Kopfer D.C. Mattfeld and C. Bierwirth. Control of parallel population dynamics by social-like behavior of ga-individuals. In *3rd PPSN*, 1994.

[6] P. Ross Fang, H.L. and D. Corne. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In *5th ICGA*, pages 375–382, 1993.

[7] J. Grabowski, E. Nowicki, and S. Zdrzalka. A block approach for single machine scheduling with release dates and due dates. *E. J. of Oper. Res.*, 26:278–285, 1986.

[8] P.J.M. van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by simulated annealing. *Oper. Res.*, 40(1):113–125, 1992.

[9] H. Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In *3rd ICGA*, pages 416–421, 1989.

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

[11] R. Nakano and T. Yamada. Conventional genetic algorithm for job shop problems. In *4th ICGA*, pages 474–479, 1991.

[12] C. R. Reeves. Genetic algorithms and neighbourhood search. In *Evolutionary Computing, AISB Workshop (Leeds, U.K.)*, pages 115–130, 1994.

[13] E.D. Taillard. Parallel taboo search techniques for the job-shop scheduling problem. *ORSA J. on Comput.*, 6(2):108–117, 1994.

[14] N.L.J. Ulder, E. Pesch, P.J.M. van Laarhoven, J. Bandelt, H, and E.H.L. Aarts. Genetic local search algorithm for the traveling salesman problem. In *1st PPSN*, pages 109–116, 1994.

[15] D. Whitley. The genitor algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In *3rd ICGA*, pages 116–121, 1989.

[16] T. Yamada and R. Nakano. A genetic algorithm applicable to large-scale job-shop problems. In *2nd PPSN*, pages 281–290, 1992.

[17] T. Yamada, B.E. Rosen, and R. Nakano. A simulated annealing approach to job shop scheduling using critical block transition operators. In *Proc. IEEE Int. Conf. on Neural Networks (Orlando, Florida.)*, pages 4687–4692, 1994.