The Fourth International Conference on Parallel Problem Solving from Nature (PPSN IV) BERLIN, GERMANY SEPTEMBER 22 - 26, 1996 pp. 960–969

Scheduling by Genetic Local Search with Multi-Step Crossover

Takeshi Yamada and Ryohei Nakano

NTT Communication Science Laboratories,
2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan

Abstract. In this paper, multi-step crossover (MSX) and a local search method are unified as a single operator called MSXF. MSX and MSXF utilize a neighborhood structure and a distance measure in the search space. In MSXF, a solution, initially set to be one of the parents, is stochastically replaced by a relatively good solution in the neighborhood, where the replacement is biased toward the other parent. After a certain number of iterations of this process, the best solution from those generated is selected as an offspring. Using job-shop scheduling problem benchmarks, MSXF was evaluated in a GA framework as a high-level crossover working on the critical path of a schedule. Experiments showed promising performance for the proposed method.

1 Introduction

It is well known that GAs are not well suited for fine-tuning structures which are very close to optimal solutions and that it is essential to incorporate local search methods, such as neighborhood search, into GAs. The result of such incorporation is often called *Genetic Local Search (GLS)* [1]. In this framework, an offspring obtained by a recombination operator, such as a crossover, is not included in the next generation directly but is used as a "seed" for the subsequent local search. The local search moves the offspring from its initial point to the nearest locally optimal point, which is included in the next generation.

In solving combinatorial optimization problems such as job-shop scheduling problems (JSSP), it is often more difficult to define a crossover operator which recombines solutions and make global changes to them than a transition operator of a neighborhood search which only modifies a solution locally. In fact, it is rather easy to construct an example of neighborhood search for JSSP by using naturally introduced job permutations and it can even be enhanced by limiting the permutations on the critical path. Unfortunately the same method cannot help build an effective crossover operator, which prevents us from applying GAs. Recently, MSX [2] was proposed as one such high-level crossover.

Given a local search method with the appropriate neighborhood structure, MSX can be defined using the same neighborhood structure in the problem space. Using the same information as the local search uses, it can locate a good, new starting point for the subsequent local search. MSX can be defined

in a problem-independent manner and can be implemented easily even if the problem to be solved is complicated.

In this paper, *Multi-Step Crossover Fusion (MSXF)*, originating from MSX, is proposed as a new high-level crossover which is fused with a local search. In MSXF, a solution, initially set to be one of the parents, is stochastically replaced by a relatively good solution from the neighborhood, where the replacement is biased toward the other parent. The biased stochastic replacement is described briefly as follows:

1. All the members in the neighborhood are indexed in ascending order according to the distance from the other parent.
2. A member is selected from the neighborhood randomly, but a smaller index is preferred. It is then probabilistically accepted according to its evaluation value.
3. If it is rejected, its index is changed to the biggest one in the neighborhood and process returns to step 2.
4. Otherwise the current solution is replaced by the selected one.

After a certain number of iterations of this process, the best one among the generated solutions is selected as an offspring. MSXF can be viewed as a recombination operator in which local search functionality is built in. In other words, it acts as a single operator unifying MSX and a local search.

MSXF has been applied to JSSP, employing a critical path-based neighborhood called the CB neighborhood. The CB neighborhood has been proved to be one of the most powerful neighborhoods for JSSP through extensive experimental studies [3]. A GA with such a tailored high-level MSXF (GA/MSXF) was evaluated with well-known benchmark problems of Muth and Thompson [4].

2 Background

2.1 Neighborhood Search

Neighborhood search is a widely used local search 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.

The outline of a neighborhood search for minimizing $V(x)$ is described in Algorithm 1, where x denotes a point in the search space and $V(x)$ denotes its evaluation value. The criterion used in Step 1 of Algorithm 1 is called the *choice criterion* by which the neighborhood search can be categorized [5]. For example, a descent method selects a point $y \in N(x)$ such that $V(y) < V(x)$. A stochastic method probabilistically selects a point according to the Metropolis Criterion, i.e., $y \in N(x)$ is selected with probability 1 if $V(y) < V(x)$; otherwise, with probability:

$$P(y) = exp(-\Delta V/T), \quad \text{where } \Delta V = V(y) - V(x) \ . \tag{1}$$

Algorithm 1. Neighborhood search
• Select a starting point: $x = x_0 = x_{best}$.
do

1. Select a point $y \in N(x)$ according to the given criterion based on the value $V(y)$. Set $x = y$.
2. If $V(x) < V(x_{best})$ then set $x_{best} = x$.

until some termination condition is satisfied.

Here P is called the acceptance probability. Simulated Annealing (SA) is a method in which parameter T (called the temperature) decreases to zero following an annealing schedule as the iteration step increases. 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.

2.2 Multi-step Crossover

In a simple GA framework, the main role of the crossover operator is to function as a search engine; by piling up good *building-blocks*, better strings can be constructed. But in GLS, local search plays the leading role and the crossover, together with the selection operator, works as a navigation engine and helps find new starting points for the subsequent local search. Multi-step crossover (MSX) is designed to be successful under such a GLS framework [2].

MSX is constructed by using the distance measure and the neighborhood structure in the search space. Let the parent solutions be p_1 and p_2, and let $N(p)$ be the neighborhood of p. Set $x = p_1$. MSX modifies x in the direction of p_2 as follows: first each point $y \in N(x)$ is ranked by the distance $d(y, p_2)$ between y and p_2; the smaller the distance, the higher the rank. Then x is replaced by y with a relatively high rank, i.e., a small $d(y, p_2)$. Repeating this step modifies x in a step-by-step manner and brings x close to p_2. In the process, x loses the characteristics of p_1 and gradually obtains those of p_2. After a certain number of iterations, the resulting new solutions contain elements of both p_1 and p_2 although in different ratios.

In combinatorial optimization problems, it is computationally effective to limit the search to a subspace of the solution space without excluding a global optimum. In a neighborhood search, this can be done by limiting the size of the neighborhood to a fraction of the total available moves, while keeping the connectivity property. The subspace is called a higher-level solution space. MSX can be viewed as a high-level crossover in GLS in the sense that a new solution generated by MSX necessarily resides in the same higher-level space as the local search uses.

3 Multi-Step Crossover Fusion

Although preliminary experiments in [2] using JSSP benchmarks demonstrated the good performance of a GA with the MSX (GA/MSX), some computational drawbacks were found. Firstly the descent method which is used as a local search method in GA/MSX is too simplistic. Secondly, a lot of individuals are generated and evaluated during the MSX steps without contributing directly toward improving the solution quality. To reduce the computational time and improve the solution quality, MSX's functionality is incorporated into a neighborhood search algorithm and these two separate operators are fused together into a single unified operator called MSXF. The neighborhood search algorithm used for the base algorithm of MSXF is not a simple decent method this time, but a more efficient stochastic one. Although SA is one of the well-known stochastic methods and has been successfully applied to many problems as well as to JSSP, it would be unrealistic to use a full SA for our purpose, because it is too time consuming to run SA many times in a GA run. A restricted method with a fixed temperature parameter $T = c$ might be a good alternative. Then the acceptance probability used in Algorithm 1 is rewritten as:

$$P_c(y) = exp\left(-\frac{\Delta V}{c}\right), \quad \text{where } \Delta V = V(y) - V(x) \quad (c : \text{constant}) \ . \tag{2}$$

MSX's functionality can be incorporated into Algorithm 1 by adding more acceptance bias in favor of $y \in N(x)$ with a small $d(y, p_2)$. The acceptance bias in MSXF is controlled by sorting $N(x)$ members in ascending order of $d(y_i, p_2)$ so that y_i with a smaller index i has a smaller distance $d(y_i, p_2)$. Here $d(y_i, p_2)$ can be estimated easily if $d(x, p_2)$ and the direction of the transition from x to y_i are known, and it is not necessary to generate and evaluate y_i. Then y_i is selected from $N(x)$ randomly, but with a bias in favor of y_i with a small index i. The outline of MSXF is described in Algorithm 2.

In place of $d(y_i, p_2)$, one can also use $\sigma(d(y_i, p_2) - d(x, p_2)) + r_\epsilon$ to sort $N(x)$ members in Algorithm 2. Here $\sigma(x)$ denotes the sign of x: $\sigma(x) = 1$ if $x > 0$, $\sigma(x) = 0$ if $x = 0$, $\sigma(x) = -1$ otherwise. A small random fraction r_ϵ is added to randomize the order of members with the same sign.

The termination condition can be given, for example, as the fixed number of iterations in the outer loop. MSXF is not applicable if the distance between p_1 and p_2 is too small compared to the number of iterations. In such a case, a mutation operator called *Multi-Step Mutation Fusion* (MSMF) is applied instead. MSMF can be defined in the same manner as MSXF except for one point: the bias is reversed, i.e., sort the $N(x)$ members in descending order of $d(y_i, p_2)$ in Algorithm 2.

4 Job-shop Scheduling and GA/MSXF

4.1 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.

Algorithm 2. Multi-Step Crossover Fusion (MSXF)

- Let p_1, p_2 be parent solutions.
- Set $x = p_1 = q$.

do
- For each member $y_i \in N(x)$, calculate $d(y_i, p_2)$.
- Sort $y_i \in N(x)$ in ascending order of $d(y_i, p_2)$.

do
1. Select y_i from $N(x)$ randomly, but with a bias in favor of y_i with a small index i.
2. Calculate $V(y_i)$ if y_i has not yet been visited.
3. Accept y_i with probability one if $V(y_i) \leq V(x)$, and with $P_c(y_i)$ otherwise.
4. Change the index of y_i from i to n, and those of y_k $(k \in \{i{+}1, i{+}2, \ldots, n\})$ from k to $k-1$.

until y_i is accepted.

- Set $x = y_i$.
- If $V(x) < V(q)$ then set $q = x$.

until some termination condition is satisfied.

- q is used for the next generation.

Each job has a technological sequence of the machines to be processed. Each *operation* requires the exclusive use of each machine for an uninterrupted duration called *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 the operations for each machine that minimizes the makespan.

Job-shop scheduling problems are not only $\mathcal{NP}$-hard , but are extremely difficult to solve optimally. To solve JSSP, exhaustive search algorithms based on branch and bound methods have been studied. Recently, approximation algorithms such as simulated annealing (SA), genetic algorithms (GAs) and tabu search have also been applied with good success [6, 7, 8, 3, 9].

Job-shop scheduling problems are often described by a disjunctive graph $G = (V, C \cup D)$, where

- V is a set of nodes representing the operations of all 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 the technological sequences of 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.

Scheduling defines the ordering between all operations that must be processed on the same machine, i.e., to fix precedences between operations. In the

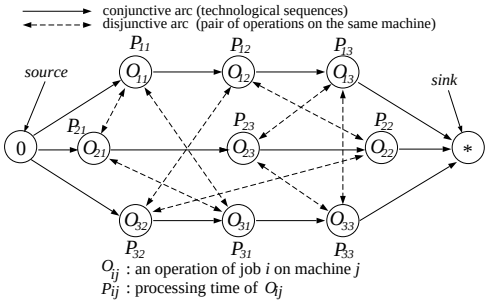

Figure 1. Disjunctive graph G of a 3×3 problem

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. In 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 two schedules S and T can be measured by the number of differences in the processing orders of operations on each machine [7]. 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 Neighborhood Structures for JSSP

A set of solutions of JSSP can be mapped to the space of bit-strings by marking each disjunctive arc as 1 or 0 according to its direction [7]. The DG distance and the Hamming distance in the mapped space are equivalent, and the neighborhood of a schedule S is a set of all (possibly infeasible) schedules whose DG distances from S are exactly one. This approach is simple and straightforward but not very efficient.

More efficient methods can be obtained by introducing a transition operator that exchanges a pair of consecutive operations only on the critical path and forms a neighborhood [10]. We call this the *adjacent swapping* (AS) neighborhood. DG distances between a schedule and members of its AS neighborhood are always one, so the AS neighborhood can be considered a subset of the bit-string neighborhood described above.

Another neighborhood using a transition operator on the critical path is proposed in [11]. The transition operator permutes the order of operations in a

critical block by moving an operation to the beginning or the end of the critical block, thus forming the *CB neighborhood*. The distances between a schedule and its CB neighbors can vary depending on the position of the moving operation.

A schedule's makespan may often be reduced by shifting an operation to left without delaying other jobs. When no such shifting can be applied to a schedule, it is called an *active schedule*. An optimal schedule is clearly active so it is safe and efficient to limit search space to the set of all active schedules. An active schedule is generated by the *GT algorithm* proposed in [12]. An extension of the CB neighborhood using the GT algorithm is proposed in [2], which is called the active CB neighborhood. It has been experimentally shown in [13] that SA using the active CB neighborhood is very powerful. Thus, the active CB neighborhood may as well be investigated in the context of GA.

4.3 GA/MSXF for Job-shop Scheduling

The MSXF is applied to JSSP using the active CB neighborhood and the DG distance previously defined. Algorithm 3 describes the outline of the GA/MSXF routine for JSSP using the steady state model proposed in [14, 15]. To avoid premature convergence even under a small-population condition, an individual whose fitness value is equal to someone in the population is not inserted to the population in step 4.

Algorithm 3. GA/MSXF for JSSP

- Initialize population: generate a set of randomly generated schedules and apply the local search to each member of the set.

do
1. Select two schedules p_1, p_2 from the population randomly with some bias depending on their makespan values.
2. **If** the DG distance between p_1, p_2 is shorter than some predefined small value, apply MSMF to p_1 and generate q.
3. **Otherwise**, apply MSXF to p_1, p_2 using the active CB neighborhood $N(p_1)$ and the DG distance, and generate a new schedule q.
4. If q's makespan is shorter than the worst in the population, and no one in the population has the same fitness value as q, replace the worst individual with q.

until some termination condition is satisfied.

- Output the best schedule in the population.

A given problem of JSSP can be converted to another problem by reversing all the technological sequences. The new problem is equivalent to the old one in the sense that, reversing the job sequences of any feasible solution from the original problem results in a feasible solution for the reversed problem with the same critical path and makespan. It can be seen, however, that an active schedule from the original problem is not necessarily active in the reversed problem. we

call a schedule *left active* if it is an active schedule for the original problem and *right active* if it is for the reversed problem. Using only the left (or right) active schedules may bias the search toward the wrong direction, therefore a mechanism to search in the space of both left and right active schedules is introduced in the GA/MSXF as follows. First, there are equal number of left and right active schedules in the initial population. The schedule q generated from p_1 and p_2 by MSXF ought to be left (or right) active if p_1 is left (or right) active, but with some probability (0.1 for example) this property is reversed.

5 Experimental Results

The performance of GA/MSXF was tested by running several simulation trials with well-known benchmark problems originated by Muth and Thompson (MT) [4].

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
	GA/MSXF	930	930	0	10	1m28s	DECα	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
	GA/MSXF	1165	1165	0	10	1m22s	DECα	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), DECα: DEC Alpha 600 5/266

Table 1 summarizes the makespan performance of the GA/MSXF method together with our previous GA/MSX method, a simulated annealing method using the CB neighborhood structure (CBSA) [3], and other GA methods published so far in literature for MT10×10 and MT20×5 problems. The GA methods include GA/GT+ECO [16], PGA+SBP [17] and GVOT [18]. For GA/MSXF, population size = 10, constant temperature $c = 10$ and the number of iterations for each MSXF = 1000 are used. The GA/MSXF experiments were performed on a DEC Alpha 600 5/226 which is about 4 times faster than a Sparc Station 10, and the programs were written in the C language. It should be noted that all of the GA/MSXF experiments successfully found optimal solutions in about one and a half minutes for both problems.

Figure 2 shows all of the solutions (in small dots) generated by an application of (a) MSXF and (b) a stochastic local search computationally equivalent to (a) for comparison. Both (a) and (b) started from the same solution (the same parent p_1), but in (a), transitions were biased toward the other solution p_2. The x-axis represents the number of disjunctive arcs whose directions are different from those of p_2 on machines with odd numbers, i.e., the DG distance was restricted to the odd machines. Similarly the y-axis representing the DG distance was restricted to the even machines.

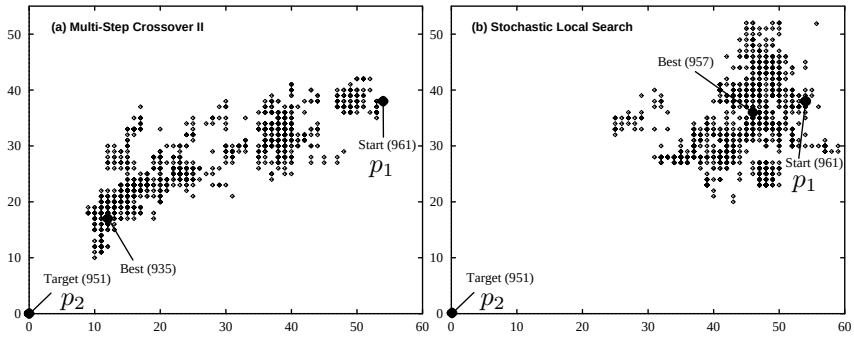

Figure 2. Distribution of solutions generated by an application of (a) MSXF and (b) a short-term stochastic local search

6 Conclusion

The multi-step crossover fusion (MSXF) has been proposed as a unified operator of a local search method and a recombination operator in GLS. MSXF uses a neighborhood structure and a measure of distance in the problem space. Starting from one parent, MSXF carries out a local neighborhood search with a limited number of iterations, where the search direction is navigated by the other parent. MSXF searches for a good solution in the problem space by concentrating its attention on the area between the parents.

We applied GA/MSXF to the job-shop scheduling problem, one of the most difficult $\mathcal{NP}$-hard combinatorial optimization problems. Preliminary experiments demonstrated that GA/MSXF outperforms GA/MSX which has been proved to be superior to other GA methods. Further research including the application of GA/MSXF to more difficult JSSP benchmark problems or to other kinds of combinatorial problems is necessary to show the full capabilities of MSXF.

Acknowledgment

Thanks to Dirk Mattfeld for his useful comments.

References

1. Ulder, N., Pesch, E., Laarhoven, P., Bandelt, H, J. and Aarts, E.: Genetic local search algorithm for the traveling salesman problem, *Proc. of 1st Problem Solving from Nature Conference (PPSN)*, pp. 109–116 (1991).
2. Yamada, T. and Nakano, R.: A GA with multi-step crossover for job-shop scheduling problems, *Proc. of Int. Conf. on GAs in Engneering Systems: Innovations and Applications (GALESIA) '95*, pp. 146–151 (1995).
3. Yamada, T., Rosen, B. and Nakano, R.: A Simulated Annealing Approach to Job Shop Scheduling using Critical Block Transition Operators, *Proc. IEEE Int. Conf. on Neural Networks (ICNN)*, pp. 4687–4692 (1994).
4. Muth, J. and Thompson, G.: *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, N.J. (1963).
5. Reeves, C. R.: Genetic Algorithms and Neighbourhood Search, *Evolutionary Computing, AISB Workshop*, pp. 115–130 (1994).
6. Aarts, E., Laarhoven, P., Lenstra, J. and Ulder, N.: A computational study of local search algorithms for job shop scheduling, *ORSA J. on Comput.*, Vol. 6, No. 2, pp. 118–125 (1994).
7. Nakano, R. and Yamada, T.: Conventional genetic algorithm for job shop problems, *Proc. of 4th Int. Conf. of Genetic Algorithms (ICGA)*, pp. 474–479 (1991).
8. Yamada, T. and Nakano, R.: A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems, *2nd PPSN*, pp. 281–290 (1992).
9. Taillard, E.: Parallel taboo search techniques for the job-shop scheduling problem, *ORSA J. on Comput.*, Vol. 6, No. 2, pp. 108–117 (1994).
10. Laarhoven, P., Aarts, E. and Lenstra, J.: Job Shop Scheduling by Simulated Annealing, *Oper. Res.*, Vol. 40, No. 1, pp. 113–125 (1992).
11. Brucker, P., Jurisch, B. and Sievers, B.: A Branch & Bound Algorithm for the Job-Shop Scheduling Problem, *Discrete Applied Mathematics*, Vol. 49, pp. 107–127 (1994).
12. Giffler, B. and Thompson, G.: Algorithms for solving production scheduling problems, *Oper. Res.*, Vol. 8, pp. 487–503 (1960).
13. Yamada, T. and Nakano, R.: Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search, *Meta-heuristics: theory & applications*, Kluwer academic publishers, MA, USA., pp. 237–248 (1996).
14. Whitley, D.: The genitor algorithm and selection pressure: why rank-based allocation of reproductive trials is best, *3rd ICGA*, pp. 116–121 (1989).
15. Syswerda, G.: Uniform crossover in genetic algorithms, *3rd ICGA*, pp. 2–9 (1989).
16. Davidor, Y., Yamada, T. and Nakano, R.: The ECOlogical Framework II: Improving GA Performance At Virtually Zero Cost, *5th ICGA*, pp. 171–176 (1993).
17. D.C. Mattfeld, H. K. and Bierwirth, C.: Control of Parallel population dynamics by social-like behavior of ga-individuals, *3rd PPSN* (1994).
18. Fang, H.L., P. R. and Corne, D.: A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems, *5th ICGA*, pp. 375–382 (1993).