IEEE International Conference on Industrial Technology (ICIT '96) Shanghai, China
DECEMBER 2 - 6, 1996 pp. 426–430

A Fusion of Crossover and Local Search

Takeshi Yamada and Ryohei Nakano
NTT Communication Science Laboratories
2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 JAPAN

Abstract— **It is well known that GAs are not well suited for fine-tuning structures that are very close to optimal solutions and that it is essential to incorporate local search methods, such as neighborhood search, into GAs. This paper explores the use of a new GA operator called multi-step crossover fusion (MSXF), which combines a crossover operator with a neighborhood search algorithm. MSXF performs a local search essentially in the region within the search space between parent solutions to find a locally optimal solution that inherits the parents' characteristics. GA/MSXF was applied to job-shop scheduling problem (JSSP). Experiments using benchmark problems show promising GA/MSXF performance even with a small population.**

I. Introduction

In solving combinatorial optimization problems such as job-shop scheduling problems (JSSP), it is often more difficult to define a crossover operator that recombines solutions and makes global changes to them than it is to define a transition operator of a neighborhood search algorithm that only modifies a solution locally. Reeves proposed a *neighborhood search crossover* for Simple GA based on an extended neighboorhood and Hamming distance[8]. In fact, it is rather easy to construct an example of neighborhood search for JSSP by using naturally introduced job permutations, and this can be further enhanced by limiting the permutations on the critical path and using active schedules. Unfortunately, the same method provides no help in building an effective crossover operator, which prevents us from applying GAs.

A GA using a binary encoding scheme was first proposed by the authors in [7]. Although this approach is simple and straightforward, it is not robust enough because some members in the binary neighborhood cannot be mapped back to job sequences or feasible schedules. Limiting the search to the set of all active schedules has proved to be more efficient. The GT crossover based on Giffler and Thompson's active scheduler algorithm was proposed as a high-level crossover in this higher-level solution space[12]. A still higher-level solution space is obtained by focusing on the critical path of a schedule and limiting the search to the critical path level. Simulated Annealing (SA) using the Critical Block (CB) neighborhood has been shown experimentally to be very powerful compared with other GA methods and SA methods using other neighborhoods[15, 14].

Therefore it is quite natural to expect that GA's population-based search strategies and the fine tuning mechanism of local neighborhood search could be unified to produce a very powerful optimization algorithm for JSSP. In this paper, *Multi-Step Crossover Fusion (MSXF)* is proposed as a new high-level crossover fused with a local search method. 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 members in the neighborhood are indexed in ascending order according to their distance from the other parent.
2. A member is randomly selected from the neighborhood, 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 largest one in the neighborhood and the 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 crossover and local search.

MSXF has been applied to JSSP, employing a critical path-based neighborhood called the CB neighborhood. Extensive experimental studies have established the CB neighborhood as one of the most powerful neighborhoods for JSSP [15]. A GA with such a tailored high-level MSXF (GA/MSXF) was evaluated with well-known benchmark problems.

II. Background

A. 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.

An 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.

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.

The criterion used in Step 1 in Algorithm 1 is called the *choice criterion*, by which the neighborhood search can be categorized. 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:

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

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.

B. Multi-Step Crossover Fusion

It is well known that GAs are not well suited for fine-tuning structures that 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)* [10]. 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.

For more complicated problems to which crossover operators are difficult to apply, this paper proposes a different approach called Multi-Step Crossover Fusion (MSXF): a new crossover operator in which local search functionality is built in. A stochastic local neighborhood search algorithm is used for the base algorithm of MSXF. Although SA is a well-known stochastic method and has been successfully applied to many problems as well as to JSSP, it would be unrealistic to apply a full SA to 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. Accordingly, the acceptance probability used in Algorithm 1 is rewritten as:

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

A crossover 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*

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 the induces 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.

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.

III. Job Shop Scheduling and GA/MSXF

The general $n \times m$ *minimum-makespan* 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 follows a technological sequence of the machines in processing. Each *operation* requires the exclusive use of one 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 each machine's operations that minimizes the makespan.

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 the 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 machine.

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

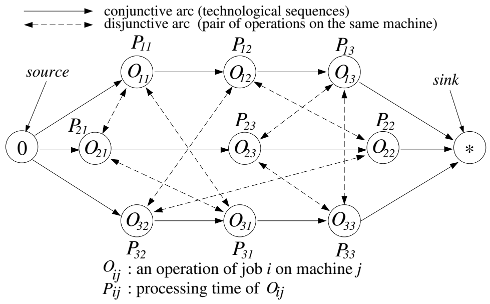

Figure 1: Disjunctive graph G of a 3×3 problem

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

A. Active schedules

A schedule's makespan may often be reduced by shifting an operation to the 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 [5].

An extension of the CB neighborhood using the GT algorithm is proposed in [13], which is called the active CB neighborhood. It was experimentally shown in [14] that SA using the active CB neighborhood is very robust.

Algorithm 3 GA/MSXF for JSSP

- Initialize population: randomly generate a set of *left* and *right* active schedules in equal number and apply the local search to each of them.

do 1. Randomly select two schedules p_1, p_2 from the population with some bias depending on their makespan values.

2. Change the direction (*left* or *right*) of p_1 by reversing the job sequences with probability P_r.
3. Do step (3a) with probability P_c, or otherwise do step (3b).
   (a) **If** the DG distance between p_1, p_2 is shorter than some predefined small value, apply MSMF to p_1 and generate q.
   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.
   (b) apply Algorithm 1 with accepting probability given by Equation 2 and active CB neighborhood.
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 an active schedule for the reversed problem. Searching only in the set of left (or right) active schedules may bias the search toward the wrong direction and result in poor local minima.

B. *GA/MSXF for Job-shop Scheduling*

The MSXF is applied to JSSP by 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 [11, 9]. 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 into the population in step 4.

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 numbers 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.

IV. Experimental Results

The performance of GA/MSXF was tested by running several simulation trials using well-known benchmark problems of Muth and Thompson (MT)[6].

Table 1 summarizes the makespan performance of the GA/MSXF method together with a simulated annealing method using the CB neighborhood structure (CBSA) [15], and performances of other GA methods published so far in the literature for MT10×10 and MT20×5 problems. The GA methods include GA/GT+ECO [2], PGA+SBP [3], GVOT [4] and GA/MSX[13]. For GA/MSXF, population size = 10, constant temperature c = 10 and number of iterations for each MSXF = 1000, $P_r = 0.1$ and $P_c = 0.5$ are used. The GA/MSXF experiments were performed on a DEC Alpha 600 5/226 which is about four times faster than a Sparc Station 10, and the programs were written in 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 each problem.

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

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	3m44s	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
	GA/MSXF	1165	1165	0	10	55s	DECα	10

Pop: population size; cpu time: average cpu time; machine: machine used for the experiments; runs: number of runs; SS2(SS10): SUN SPARC Station 2(10); DECα: DEC Alpha 600 5/266

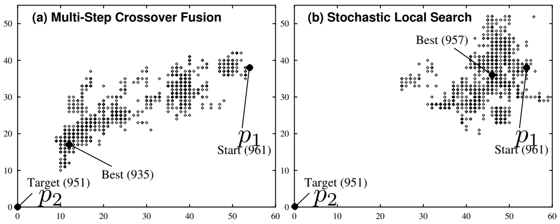

Figure 2: Distribution of solutions generated by an application of (a) MSXF and (b) a short-term stochastic local search

Table 2: Results of 10 touch JSSPs

prob	lb	bst	avg	var	wst	n_{opt}	t_{opt}
abz7	655	678	692.5	0.94	703	–	–
abz8	638	686	703.1	1.54	724	–	–
abz9	656	697	719.6	1.53	732	–	–
la21	–	*1046	1049.9	0.57	1055	9	687.7
la24	–	*935	938.8	0.34	941	4	864.1
la25	–	*977	979.6	0.40	984	9	765.6
la27	–	*1235	1253.6	1.56	1269	1	2364.75
la29	1130	1166	1181.9	1.31	1195	–	–
la38	–	*1196	1198.4	0.71	1208	21	1051.3
la40	*1222	1224	1227.9	0.43	1233	–	–

restricted to the odd machines. Similarly, the y-axis representing the DG distance was restricted to the even machines.

Table 2 shows the makespan performances of GA/MSXF for the ten difficult benchmark JSSPs[1]. The parameters used here are the same as those of the MT benchmark except for population size = 20. The algorithm was terminated when an optimal solution was found or after 40 minutes of cpu time on the DEC Alpha 600 5/266. In the table, the column named lb shows the known lower bound or known optimal value (for la40) of makespan, and the columns named bst, avg and wst show the best, average, variance and worst makespan values obtained, respectively. The columns named n_{opt} and t_{opt} show the number of runs in which the optimal schedules are obtained and their average cpu times.

The optimal solutions were found for half of the ten problems, and four of them were found very quickly. The small variances in the solution qualities indicate the stableness of GA/MSXF as an approximation method.

V. Conclusion

The multi-step crossover fusion (MSXF) is proposed as a unified operator of a local search method and a recombination operator in GLS. MSXF uses a neighborhood structure and a distance measure 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. Experiments using Muth and Thompson's benchmark demonstrated that GA/MSXF outperforms other GA methods described in the literature. GA/MSXF could find near-optimal solutions for ten difficult benchmark problems, including optimal solutions for five problems.

Further research, including the application of GA/MSXF to other kinds of combinatorial problems, is necessary to show the full capabilities of MSXF.

References

[1] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA J. on Comput.*, 3(2):149–156, 1991.

[2] 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.

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

[4] 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.

[5] B. Giffler and G.L. Thompson. Algorithms for solving production scheduling problems. *Oper. Res.*, 8:487–503, 1960.

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

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

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

[9] G. Syswerda. Uniform crossover in genetic algorithms. In *3rd ICGA*, pages 2–9, 1989.

[10] 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.

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

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

[13] T. Yamada and R. Nakano. A genetic algorithm with multi-step crossover for job-shop scheduling problems. In *Proc. of Int. Conf. on GAs in Eng. Sys. (GALESIA) '95*, pages 146–151, 1995.

[14] T. Yamada and R. Nakano. *Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search.* Kluwer academic publishers, MA, USA, 1996.

[15] 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.