Genetic and Local Search Algorithms as Robust and Simple Optimization Tools

Mutsunori Yagiura and Toshihide Ibaraki
Department of Applied Mathematics and Physics
Graduate School of Engineering
Kyoto University
Kyoto-city, 606-01 Japan
Email: yagiura, ibaraki@kuamp.kyoto-u.ac.jp

Abstract:
One of the attractive features of recent metaheuristics is in its robustness and simplicity. To investigate this direction, the single machine scheduling problem is solved by various genetic algorithms (GA) and random multi-start local search algorithms (MLS), using rather simple definitions of neighbors, mutations and crossovers. The results indicate that: (1) the performance of GA is not sensitive about crossovers if implemented with mutations, (2) simple implementation of MLS is usually competitive with (or even better than) GA, (3) GRASP type modification of MLS improves its performance to some extent, and (4) GA combined with local search is quite effective if longer computational time is allowed.

Key Words: combinatorial optimization, metaheuristics, genetic algorithm, local search, GRASP, single machine scheduling.

1. Introduction

There are numerous combinatorial optimization problems, for which computing exact optimal solutions is computationally intractable, e.g., those known as *NP-hard* [6]. However, in practice, we are usually satisfied with good solutions, and in this sense, *approxi-*

mate (or *heuristic*) *algorithms* are very important. In addition to common heuristic methods such as *greedy method* and *local search* (abbreviated as LS), there is a recent trend of *metaheuristics* which combine such heuristic tools into more sophisticated frameworks. Those metaheuristics include *random multi-start local search* (abbreviated as MLS) [4][15], *simulated annealing*, *tabu search*, *genetic algorithm* (abbreviated as GA) [3][8][10] and so on.

One of the attractive features of these metaheuristics is in its simplicity and robustness. They can be developed even if deep mathematical properties of the problem domain are not at hand, and still can provide reasonably good solutions, much better than those obtainable by simple heuristics. We pursue this direction more carefully in this paper, by implementing various metaheuristics such as MLS and GA, and comparing their performance. The objective of this paper is not to propose the most powerful algorithm but to compare general tendencies of various algorithms. The emphasis is placed not to make each ingredient of such metaheuristics too sophisticated, and to avoid detailed tuning of the program parameters involved therein, so that practitioners can easily test the proposed framework to solve their problems of applications.

The LS starts from an initial solution x and repeats replacing x with a better solution in its *neighborhood* $N(x)$ until no better solution is found in $N(x)$. The MLS applies the LS to a number of randomly generated initial solutions and outputs the best solution found during the entire search. A variant of MLS called GRASP (greedy randomized adaptive search procedure) differs from MLS in that it generates initial solutions by randomized greedy methods. Some good computational results are reported in [4][5][14]. Simulated annealing and tabu search try to enhance the local search by allowing the replacement of the current solution x with a worse solution in $N(x)$ thereby avoiding to be trapped into bad local optima. The GA is a probabilistic algorithm that simulates the evolution process, by improving a set of candidate solutions as a result of repeating the operations such as crossover, mutation and selection. Among various modifications [3][20][23], it is reported in [1][13][16][22] that introducing local search technique in GA is quite effective. This is called *genetic local search* (abbreviated as GLS).

As a concrete problem to test, we solve in this paper the single machine scheduling problem (abbreviated as SMP), which is known to be NP-hard. It asks to determine an optimal sequence of n jobs that minimizes a cost function defined for jobs, e.g., the total weighted sum of earliness and tardiness. The efficiency of algorithms are measured on the basis of the number of solution samples evaluated, rather than the computational time, since the computational time depends on the computers used and other factors such as programming skill.

Our computational results indicate that GA performs rather poorly if mutation is not used, and is very sensitive to the crossover operations employed. However, if mutation is introduced, much better solutions are obtained almost independently of the crossover operators in use. Therefore, GA, using both crossover and mutation operators, is not a bad choice for practical purposes. However, a simple implementation of MLS appears better since it usually gives solutions of similar (or even better) quality with the same number of samples. Finally, it is also observed that GLS and GRASP tend to outperform GA and MLS in the sense that they produce solutions of better quality if sufficiently large number of samples are tested.

Our experiment does not cover other metaheuristics such as simulated annealing and tabu search, and it is a target of our future research to assess and compare such metaheuristics.

2. Single Machine Scheduling Problem

The single machine scheduling problem (SMP) asks to determine an optimal sequence of n jobs in $V = \{1, \ldots, n\}$, which are processed on a single machine without idle time. A sequence σ: $\{1, \ldots, n\} \rightarrow V$ is a one-to-one mapping such that $\sigma(k) = j$ means that j is the k-th job processed on the machine. Each job i becomes available at time 0, requires integer processing time p_i and incurs cost $g_i(c_i)$ if completed at time c_i, where $c_i = \sum_{j=1}^{\sigma^{-1}(i)} p_{\sigma(j)}$. A sequence σ is optimal if it minimizes

$$\mathrm{cost}(\sigma) = \sum_{i \in V} g_i(c_i). \tag{1}$$

The SMP is known to be NP-hard for most of the interesting forms of $g_i(\cdot)$. We consider in particular

$$g_i(c_i) = h_i \max\{d_i - c_i, 0\} + w_i \max\{0, c_i - d_i\}, \tag{2}$$

where $d_i \in Z_+$ (set of nonnegative integers) is the due date of job i, and $h_i, w_i \in Z_+$ are respectively the weights given to earliness and tardiness of job i.

A number of exact algorithms, which are based on branch-and-bound, have been studied so far [19]. Another type of algorithm *SSDP* (*successive sublimation dynamic programming*) was proposed in [12] and it is observed that problem instances of up to $n = 35$ can be practically solved to optimality.

3. Metaheuristic Algorithms

3.1 Local Search and Multi-Start Local Search

The local search (LS) proceeds as follows, where $N(\sigma)$ denotes the neighborhood of σ.

LOCAL SEARCH (LS)

1 (initialize): Generate an initial solution σ by using some heuristics.

2 (improve): If there exists a solution $\sigma' \in N(\sigma)$ satisfying $\text{cost}(\sigma') < \text{cost}(\sigma)$, then set $\sigma := \sigma'$ and repeat step 2; otherwise output σ and stop.

A solution σ satisfying $\text{cost}(\sigma) \leq \text{cost}(\sigma')$ for all $\sigma' \in N(\sigma)$ is called a *local optimal solution.*

The LS may be enhanced by repeating it from a number of randomly generated initial solutions and outputs the best solution found during the entire search. This is called the *random multi-start local search* (MLS).

The performance of LS and MLS critically depends on: (1) the way of generating initial solutions, (2) the definition of $N(\sigma)$, and (3) the search strategy (i.e., how to search the solutions in $N(\sigma)$). In our experiment, we examine only the following strategies from the view point of simplicity.

(1) Initial solutions: σ are randomly selected with equal probability.

(2) Neighborhoods: $N_{ins}(\sigma) = \{\sigma_{i \leftarrow j} \mid i \neq j\}$ and $N_{swap}(\sigma) = \{\sigma_{i \leftrightarrow j} \mid i \neq j\}$. Here $\sigma_{i \leftarrow j}$ is the sequence obtained from σ by moving the j-th job to the location before the i-th job, while $\sigma_{i \leftrightarrow j}$ is obtained by interchanging the i-th job and j-th job of σ.

(3) Search strategies: FIRST scans $N(\sigma)$ according to a prespecified random order and selects the first improved solution σ' satisfying $\text{cost}(\sigma') < \text{cost}(\sigma)$, and BEST selects the solution σ' having the best cost in the entire area of $N(\sigma)$.

Many other neighborhood structures are possible (e.g., [15] for the traveling salesman problem). Other metaheuristics such as simulated annealing and tabu search can be considered as modifications of MLS in which sophisticated search strategies for $N(\sigma)$ are used.

3.2 Greedy Randomized Adaptive Search Procedure

Greedy randomized adaptive search procedure (GRASP) is a variation of MLS [4][5][14], in which the initial solutions are generated by randomized greedy methods. In our experiment, the initial solutions are generated as follows. At each step, let

$$M = \{i \in V \mid i \text{ is not scheduled yet}\}.$$

Then a job $i \in M$ is chosen as the next job according to a criterion based on a local gain function (e.g., the i with the smallest d_i). There are two types of algorithms, depending upon whether a schedule is constructed forward or backward. A forward schedule starts with the job to be processed at time 0, and continues adding the next job to be processed until M becomes empty. A backward schedule is symmetrically defined from the last job to the first job. We will describe below the forward schedule only.

1. Set $M := V$ and $t := 0$.
2. Choose a candidate set $CA \subseteq M$ of a fixed number of jobs ($|CA|$ is a prespecified positive integer) in the decreasing order of the local gain $e(i,t)$, and then randomly choose $i \in CA$ as the next job. Let $M := M - \{i\}$ and $t := t + p_i$.
3. Repeat step 2 until $M = \phi$.

Here the local gain function $e(i,t)$ represents the heuristic used. We employed the following six functions (and hence 12 algorithms corresponding to forward and backward constructions).

The first three local gain functions are given by

$$e_1(i,t) = g_i(t+p_i+\bar{p})\delta_i(t) - g_i(t+p_i)(1-\delta_i(t)) \quad (3)$$

$$e_2(i,t) = w_i\delta_i(t) - h_i(1-\delta_i(t)) \quad (4)$$

$$e_3(i,t) = \frac{w_i}{p_i}\delta_i(t) - \frac{h_i}{p_i}(1-\delta_i(t)), \quad (5)$$

where $\bar{p}$ is the average processing time $\bar{p} = \sum_{i \in V} p_i/n$, and $\delta_i(t)$ indicates whether the due date d_i is urgent or not, i.e.,

$$\delta_i(t) = \begin{cases} 1, & t+\bar{p}+p_i \geq d_i, \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

To define $e_4(i,t)$, suppose that all jobs j in $M-\{i\}$ are sorted in nondecreasing order of d_j, i.e., $d_{j_1} \leq \ldots \leq d_{j_k}$. Then

$$e_4(i,t) = -g_i(t+p_i) - \sum_l g_{j_l}(c_{j_l}), \quad (7)$$

where $c_{j_l} = t + p_i + \sum_{h=1}^{l} p_{j_h}$. This gives the sum of $g_j(c_j)$ of all jobs in M if i is scheduled next and the rest is then scheduled in nondecreasing order of d_j. Functions e_5 and e_6 are similar to e_4 but use nonincreasing order of w_j and nondecreasing order of h_j, respectively, for set $M-\{i\}$.

3.3 Genetic Algorithm

The genetic algorithm (GA) is a probabilistic algorithm that repeats the operations such as crossover, mutation and selection to the set of candidate solutions P. This algorithm can be viewed as a generalization of LS, in which the neighborhood is defined to be the set of solutions obtainable from P by crossover and mutation operators. We denote such neighborhood by $N_{CROSS-MUT}$ if crossover operator CROSS and mutation operator MUT are employed.

GENETIC ALGORITHM (GA)

1 (initialize): Construct a set of initial candidate solutions P.

2 (crossover and mutate): Generate a set of solutions $Q \subseteq N_{CROSS-MUT}(P)$.

3 (select): Select a set of $|P|$ solutions P' from $P \cup Q$, and set $P := P'$.

4 (iterate): Repeat steps 2 to 3 until some stopping criterion is satisfied.

Among various types of crossover, mutation and selection operators known so far [3][8][17][20][24], we consider representative operators, as described in the following Subsections.

3.3.1 Crossover

A crossover operator generates one or more solutions (*children*) by combining two or more candidate solutions (*parents*). Here, we assume that one child σ_C is produced from two parents σ_A and σ_B.

PMX (partially mapped crossover)[7]: An n-bit mask $m \in \{0,1\}^n$ is generated randomly. Set $\sigma_C(i) := \sigma_A(i)$ for all i with $m(i) = 0$, and for each i with $m(i) = 0$ exchange $\sigma_B(i)$ with $\sigma_B(j)$ such that $\sigma_B(j) = \sigma_A(i)$. Then set $\sigma_C(i) := \sigma_B(i)$ for all i with $m(i) = 1$.

Crossover operators based on arbitrary masks are called uniform, and those based on restricted masks having at most k adjacent 0-1 pairs are called k-point; e.g., masks 11000 and 10011 are 1-point and 2-point respectively. Here we consider 1-point, 2-point and uniform PMX, which are respectively denoted PMX1, PMX2 and PMXU.

CX (cycle crossover)[17]: From sequences σ_A and σ_B, form a permutation $\binom{\sigma_A}{\sigma_B}$, and decompose it into cycles $R_1, R_2, \ldots, R_{k_0}$, where R_k denotes the set of positions i in the k-th cycle. By definition, $\{\sigma_A(i) \mid i \in R_k\} = \{\sigma_B(i) \mid i \in R_k\}$ holds for all k. Then partition set $K = \{1,2,\ldots,k_0\}$ into K_A and K_B, and define σ_C by setting $\sigma_C(i) := \sigma_A(i)$ for all $i \in R_k$, if $k \in K_A$, and $\sigma_C(i) := \sigma_B(i)$ for all $i \in R_k$, if $k \in K_B$. Various strategies for partitioning K are possible. Here we consider (1) CXU: K is randomly partitioned into K_A and K_B, (2) CX1: $k \in K$ is randomly chosen, and $K_A := \{k\}$, $K_B := K - K_A$, and (3) CXA: $K_A := \{1,3,5,\ldots\}$ and $K_B := \{2,4,\ldots\}$.

PsRND (position-based random crossover)[24]: Represent σ_A by a set of pairs $T_A = \{(i, \sigma_A(i)) \mid i = 1, \ldots, n\}$; similarly for T_B and T_C. A child σ_C is constructed by starting from $T_C := \emptyset$ and by repeating the following as far as $T_A \cup T_B \neq \emptyset$ holds.

> Randomly select an element $e \in T_A \cup T_B$ and set $T_C := T_C \cup \{e\}$, and remove the elements in $T_A \cup T_B$ which contradict with T_C.

If this halts before completing T_C by $T_A \cup T_B = \emptyset$, then complete T_C by adding the noncontradicting elements to T_C.

OX (order crossover)[2][16]: Generate randomly an n-bit mask $m \in \{0,1\}^n$. Set $\sigma_C(i) := \sigma_A(i)$ for all i satisfying $m(i) = 0$. Define $D'_B = D_B - \{(i,j) \mid i \in S_m \text{ or } j \in S_m\}$, where $D_B = \{(i,j) \mid \sigma_B^{-1}(i) \leq \sigma_B^{-1}(j)\}$ and $S_m = \{\sigma_A(i) \mid m(i) = 0\}$. Then D'_B is the total order of σ_B restricted to $V - S_m$. Complete σ_C by assigning jobs to all the positions i satisfying $m(i) = 1$ according to D'_B. We call 1-point, 2-point and uniform crossover operators of this type as OX1, OX2 and OXU respectively.

Others: Many other crossover operators for sequencing problems have been proposed, e.g., free list crossover [9], partial order preserving crossover [24], alternating edges crossover [9], edge recombination crossover [20][23], matrix crossover [11], simulated crossover [1][21], and so on. Several of them were examined in [24] under a simple framework of GA without mutations.

3.3.2 Mutation

Mutation employed in this paper perturbs a candidate solution σ by repeating i times the random selection $\sigma' \in N(\sigma)$ and $\sigma := \sigma'$, where $i \in [1, k]$ is a positive integer randomly chosen, and k is a program parameter. Two types of neighborhood $N_{ins}(\sigma)$ and $N_{swap}(\sigma)$ are used as $N(\sigma)$; the resulting mutations are denoted as INSk and SWAPk respectively.

3.3.3 Generation of Candidate Solutions

Even after the crossover and mutation operators are fixed (i.e., $N_{CROSS-MUT}$ is defined), the set of generated candidate solutions $Q \subseteq N_{CROSS-MUT}(P)$ in step 2 of GA depends on other details. In

our experiment, we use $|Q| = 1$ and the child $\sigma_C \in Q$ is generated as follows.

1. Randomly select two parents σ_A, $\sigma_B \in P$.
2. After mutating either σ_A or σ_B by operator MUT, crossover them to produce σ_C by operator CROSS.

3.3.4 Selection

Selection in step 3 of GA chooses good candidate solutions from set $P \cup Q$ for the next generation. Though various types of selection strategies have been proposed [3][8], here we employ the following two simple strategies. The selection is executed only if the child $\sigma_C \in Q$ is not in P.

(1) BESTP: With a solution σ_{worst} satisfying $\mathrm{cost}(\sigma_{worst}) \geq \mathrm{cost}(\sigma)$ for all $\sigma \in P \cup Q$, let $P' := P \cup \{\sigma_C\} - \{\sigma_{worst}\}$.

(2) DELWP: Suppose $\mathrm{cost}(\sigma_A) \leq \mathrm{cost}(\sigma_B)$ without loss of generality, and let $P' := P \cup \{\sigma_C\} - \{\sigma_B\}$ with probability

$$p := \begin{cases} 1, & \text{if } \mathrm{cost}(\sigma_C) \leq \mathrm{cost}(\sigma_B) \\ \frac{\Delta_B}{\Delta_B + \Delta_C}, & \text{otherwise,} \end{cases}$$

where $\Delta_B := \mathrm{cost}(\sigma_B) - \mathrm{cost}(\sigma_A)$, $\Delta_C := \mathrm{cost}(\sigma_C) - \mathrm{cost}(\sigma_A)$; $P' := P$ with probability $1 - p$.

3.4 Genetic Local Search

Genetic local search (GLS) is a variation of GA [1][13][16][22], in which the new candidate solutions in Q are improved by LS.

GENETIC LOCAL SEARCH (GLS) (Steps 1 to 4 are the same as GA. Step $2'$ is added between steps 2 and 3 of GA.)

$2'$ (improve): Improve each solution $\sigma \in Q$ by applying local search (LS).

GLS is different from MLS in that GLS generates the initial solutions for LS from the current P by crossover and/or mutation, while MLS generates them randomly from scratch.

While the improvement of a solution σ can be continued until a local optimal solution is obtained, cutting off the search at certain point is also possible to save computation time, since most of the

improvements usually occur in the early stage of LS. We examine the following simple termination rule: the number of samples for each LS is bounded above by $b|N(\sigma)|$ (b is a given positive integer, where $b = \infty$ means the original LS).

4. Computational Results

Computational experiments were performed on SUN SPARC station IPX using C language. The quality of the obtained solutions is evaluated by the average error from the best cost for the same problem instance found during the entire experiment.

4.1 Generation of Problem Instances

The tested problem instances are generated as follows. For $n = 35$ and 100, coefficients p_i, h_i, w_i for $i \in V$ $(= \{1, \ldots, n\})$ are generated by randomly selecting integers from interval [1, 10]. It has been observed in the literature (e.g., [18]) that problem hardness is related to two parameters RDD and LF, called the relative range of due dates and the average lateness factor, respectively. In our experiment,

$$\begin{aligned} RDD &= 0.2, 0.4, 0.6, 0.8, 1.0, \\ LF &= 0.2, 0.4, \end{aligned}$$

are used. Corresponding to each of these $5 \times 2 = 10$ cases, one problem instance is generated by selecting integer due dates d_i, $i \in V$, from interval

$$[(1 - LF - RDD/2)MS,\ (1 - LF + RDD/2)MS],$$

where $MS = \sum_{i \in V} p_i$. Sizes $n = 35$ and 100 are chosen, since $n = 35$ is the largest size that were solved to optimality by the SSDP method [12], and larger sizes should also be tested.

4.2 Random Multi-Start Local Search

Two types of neighborhoods N_{ins} and N_{swap}, and two types of search strategies FIRST and BEST of Section 3.1 are all examined. The average error (%) of the best solutions obtained by these four combinations are shown in Table 1, where 3×10^5 and 3×10^6 samples are generated for $n = 35$ and 100, respectively. Table 1 also shows

Table 1: Average error in % of the best solutions (average number of initial solutions) with MLS.

	$n = 35$; 3×10^5 samples		$n = 100$; 3×10^6 samples	
	N_{ins}	N_{swap}	N_{ins}	N_{swap}
FIRST	0.000 (97.6)	0.000 (103.6)	0.660 (103.6)	0.194 (104.6)
BEST	0.289 (8.6)	0.047 (13.9)	4.694 (3.2)	0.622 (5.0)

Table 2: Average error in % (standard deviation) of 100 local optima.

	$n = 35$		$n = 100$	
	N_{ins}	N_{swap}	N_{ins}	N_{swap}
FIRST	4.246 (4.670)	2.653 (4.078)	2.484 (2.332)	1.303 (1.491)
BEST	6.548 (7.473)	2.263 (3.582)	6.204 (5.475)	1.551 (1.699)

the average number of trials (i.e., the number of initial solutions) in parentheses. Note that even if a trial of LS is not completed at the time of termination, such a trial is also counted as one.

These results indicate that: (1) Search strategy FIRST obtains good solutions earlier than search strategy BEST. (Based on this, the search strategy is fixed to FIRST in the following experiments.) (2) The quality of solutions obtained by neighborhood N_{swap} is slightly better than that obtained by N_{ins}.

We also examined the average quality of local optimal solutions obtained by these local search algorithms. Table 2 shows the average error in % (standard deviation) of 100 (10 trials for each of 10 instances) local optima generated by each of the four combinations.

We observe that: (1) The differences of the solution quality among two search strategies are not critical. (2) The solution quality of N_{swap} local optima is better than that of N_{ins} on the average. (3) The combination of N_{ins} and BEST produces solutions of rather poor quality.

These results indicate that FIRST allows much more number of local optima being attained, where solution quality of each local optimum is comparable with that obtained by using BEST, within the same number of solution samples. This is the reason why FIRST

Table 3: Average error in % of the best solutions with GRASP using N_{swap}. The last column, init, indicates the average error of the initial greedy solutions generated by conventional greedy methods.

$\|CA\|$	1	2	4	7	10	20	init
e_1 (f)	0.181	0.154	0.226	0.168	0.167	0.149	50.5
e_1 (b)	0.203	0.306	0.191	0.221	0.228	0.152	53.8
e_2 (f)	1.522	1.290	1.031	0.651	0.443	0.212	28.6
e_2 (b)	1.337	1.115	0.648	0.421	0.268	0.148	33.6
e_3 (f)	0.381	0.333	0.200	0.134	0.075	0.161	8.8
e_3 (b)	0.566	0.396	0.191	0.218	0.135	0.180	16.0
e_4 (f)	0.168	0.097	0.118	0.095	0.151	0.186	18.2
e_4 (b)	0.464	0.390	0.212	0.306	0.202	0.239	72.7
e_5 (f)	0.118	0.169	0.115	0.145	0.067	0.143	73.2
e_5 (b)	0.136	0.107	0.120	0.117	0.093	0.129	90.6
e_6 (f)	0.170	0.133	0.053	0.104	0.107	0.105	116.0
e_6 (b)	0.156	0.120	0.126	0.133	0.086	0.136	64.3
MLS	0.194						314.6

shows better performance in the computational results of Table 1.

4.3 Greedy Randomized Adaptive Search Procedure

A total of 12 local gain functions and parameter values $|CA| = 1, 2, 4, 7, 10, 20$ are tested to generate initial solutions of GRASP, where $|CA| = 1$ means the conventional greedy methods. In this Subsection, only the results with the neighborhood N_{swap} are shown; however, similar tendencies were observed for N_{ins}. Table 3 shows the average error (%) of the best solutions obtained by various GRASP algorithms, where e_i (f) (resp., e_i (b)) means the forward (resp., backward) construction with local gain function e_i. For the comparison purpose, the last row, MLS, indicates the average error when initial solutions are generated randomly. The last column, init, indicates the average error of the solutions generated by greedy methods, which means the quality of initial solutions used by GRASP when $|CA| = 1$.

The results indicate that: (1) Performance of GRASP critically depends on the local gain functions used for generating initial so-

lutions. (2) If the local gain function is properly chosen, GRASP improves the performance of MLS to some extent. However the performance is hardly affected by the parameter $|CA|$. (3) A local gain function which produces solutions of better quality does not always lead to better performance of GRASP.

In other words, GRASP is simple and can be powerful than MLS, but not robust with the local gain function used.

Table 4: Average error (%) of the best solutions with various versions of GA with $|P| = 100$.

		$n = 35$; 3×10^4 samples			$n = 100$; 3×10^5 samples		
		no MUT	INS1	SWAP1	no MUT	INS1	SWAP1
	CXA	46.5	1.1	1.2	97.7	2.8	1.1
	CX1	46.4	1.8	2.4	96.2	2.3	0.8
	CXU	61.9	0.6	1.5	100.0	3.7	1.1
B	PMX1	54.4	2.2	2.2	136.3	2.4	1.1
E	PMX2	16.3	0.1	0.6	47.2	1.9	1.1
S	PMXU	8.0	1.8	0.8	27.7	1.6	1.2
T	PsRND	9.4	0.06	0.4	25.9	1.4	1.3
P	OX1	110.1	5.1	3.5	190.9	3.1	2.1
	OX2	10.5	0.2	0.8	35.9	2.6	0.7
	OXU	1.4	0.3	1.1	3.8	0.8	0.8
	MUT only	—	2.3 ($0.7^{\dagger}$)	4.0 ($1.1^{\dagger}$)	—	1.4 ($1.2^{\ddagger}$)	1.5 ($0.7^{\ddagger}$)
	CXA	48.2	1.3	0.8	83.4	3.3	0.8
	CX1	43.2	1.1	4.3	90.8	2.5	2.0
	CXU	40.8	1.6	2.1	91.5	4.0	0.7
D	PMX1	57.5	1.3	1.3	136.1	2.1	1.5
E	PMX2	21.2	1.0	0.7	57.1	1.4	1.6
L	PMXU	4.8	0.6	1.0	22.5	1.8	0.8
W	PsRND	4.9	0.3	1.4	20.3	1.2	1.1
P	OX1	107.9	4.2	2.6	184.3	3.1	1.8
	OX2	10.3	0.8	0.2	36.2	2.1	0.3
	OXU	0.9	0.07	0.1	2.4	1.1	0.5
	MUT only	—	3.3 ($1.1^{\dagger}$)	4.8 ($0.4^{\dagger}$)	—	1.4 ($0.9^{\ddagger}$)	1.9 ($0.7^{\ddagger}$)
	MLS	—	0.1	0.06	—	1.0	0.5

$\dagger$ results after 10^5 samples. $\ddagger$ results after 10^6 samples.

4.4 Genetic Algorithm

Various neighborhoods $N_{CROSS-MUT}$ are compared under the framework of GA, in which $|P|$ is set to 100 and two selection strategies BESTP and DELWP are examined. Note that exactly one cost

evaluation occurs during one generation, since $|Q| = 1$ is used. Table 4 shows the average error (%) of the best solutions obtained by the tested algorithms, where 3×10^4 (3×10^5) samples are allowed for $n = 35$ ($n = 100$). Here, 'MUT only' means that crossover is not used and 'no MUT' means that mutation is not used. The results of MLS are also included for comparison, where the same neighborhood as mutation is used.

These results indicate that: (1) Using mutations is essential to get good solutions within the framework of GA. (2) Performance of GA is not sensitive against the types of crossover operators if combined with mutation, though it critically depends on the types of crossover operators if mutation is not used. (3) Selection strategy has little effect on the performance of GA. (4) Crossover is also effective to improve GA, since GA with MUT only needs slightly more samples than GA with crossover to obtain solutions of similar quality. (5) MLS performs better than GA.

We also tested GA when mutation rate k of Section 3.3.2 is set to greater than one. The results are omitted because of the limited space. It is observed that the performance of GA is hardly affected by parameter k if combined with crossover operators. However, the convergence of GA with MUT only becomes slower as k increases. It is also observed that $k = 1$ is sufficient for obtaining solutions of good quality.

Other population sizes $|P| = 10$ and 1000 are also tested so that we can compare the effect of increasing $|P|$ and that of incorporating mutation. The results are shown in Table 5. The search is terminated when no more improvement is considered to occur; to be concrete, 300 samples are tested for $|P| = 10$, and 2×10^5 (resp., 9×10^5) samples for $n = 35$ (resp., $n = 100$) are tested for $|P| = 1000$. Selection strategy BESTP is used, but mutation is not incorporated. Table 5 also include the results for $|P| = 100$ with mutation, where the same number of samples as Table 4 are tested. The solution quality is improved on the average if a larger $|P|$ is used. However, it critically depends on the crossover operator. In general, much more number of samples appears to be required to obtain good solutions by only increasing the population $|P|$, and mutation appears necessary to add.

Table 5: Average error (%) of the best solutions with GA using $|P| = 10, 100, 1000$.

	$n = 35$				$n = 100$			
$\|P\|$	10	100		1000	10	100		1000
mutation	noMUT	noMUT	SWAP1	noMUT	noMUT	noMUT	SWAP1	noMUT
CXA	237.8	46.5	1.2	2.5	262.9	97.7	1.1	10.6
CX1	248.7	46.4	2.4	2.8	271.7	96.2	0.8	13.8
CXU	221.4	61.9	1.5	3.5	250.3	100.0	1.1	12.5
PMX1	221.4	54.4	2.2	4.0	260.6	136.3	1.1	31.2
PMX2	228.0	16.3	0.6	0.4	242.6	47.2	1.1	9.1
PMXU	137.1	8.0	0.8	0.5	233.3	27.7	1.2	31.0
PsRND	165.1	9.4	0.4	0.2	190.1	25.9	1.3	31.4
OX1	251.1	110.1	3.5	16.7	292.8	190.9	2.1	94.3
OX2	222.3	10.5	0.8	0.3	274.7	35.9	0.7	4.6
OXU	173.6	1.4	1.1	0.03	208.3	3.8	0.8	0.5

Table 6: Average error in % of the best solutions with GA after 3×10^6 samples.

$\|P\|$	10^1	10^2	10^3	10^4	MLS
INS1	4.874	0.639	0.593	0.522	0.660
SWAP1	0.802	0.800	0.536	0.325	0.194

GA with more tested samples is also tested. Table 6 shows the average error in % of the best solutions after 3×10^6 samples are generated, where $n = 100$, the selection strategy is BESTP, crossover type is OXU. For comparison purpose, the results of MLS are also included in Table 6, where the same neighborhood as the mutation is used.

The results indicate that: (1) Better solutions are obtained on the average as $|P|$ increases at the cost of testing more number of samples. (2) The quality of solutions obtained by MLS is still slightly better than those results of GA.

Note that much more computational time is needed to sample a solution with GA compared to other algorithms, such as MLS. We may conclude that the effectiveness of simple GA is in question.

Table 7: Average error (%) of the best solutions with GLS in which $|P| = 20$ and BESTP is used.

bound	$b = \infty$						$b = 1$	
neighbor	N_{ins}			N_{swap}			N_{ins}	N_{swap}
mutation	noMUT	INS1	SWAP1	noMUT	INS1	SWAP1	noMUT	noMUT
CXA	0.256	0.193	0.266	0.077	0.029	0.059	0.332	0.081
CX1	0.709	0.621	0.595	0.318	0.122	0.163	0.596	0.277
CXU	0.243	0.233	0.274	0.082	0.048	0.060	0.260	0.074
PMX1	0.286	0.315	0.335	0.092	0.062	0.081	0.327	0.157
PMX2	0.312	0.249	0.297	0.041	0.040	0 032	0.190	0.071
PMXU	0.264	0.271	0.341	0.028	0.023	0.046	0.205	0.028
PsRND	0.265	0.403	0.391	0.018	0.067	0.061	0.107	0.249
OX1	0.279	0.359	0.301	0.088	0.062	0.067	0 384	0.272
OX2	0.218	0.325	0.270	0.036	0.049	0.045	0.120	0.053
OXU	0.201	0.162	0.264	0.017	0.067	0.036	0.095	0.027
MUT only	—	0.596	0.508	—	0.100	0.054	—	—
MLS	0.660			0.194				

4.5 Genetic Local Search

It is observed by preliminary experiments that, within 3×10^5 (3×10^6) samples for $n = 35$ ($n = 100$), roughly 100 independent trials of LS starting from random initial solutions are possible using either N_{ins} or N_{swap}. $|P| = 20$ and selection strategy BESTP are used, and $b = \infty$ is assumed unless otherwise stated. By using these parameters, almost all the tested algorithms could obtain exact optimal solutions for all the instances of $n = 35$, and hence the results for $n = 35$ are omitted. Table 7 shows the average error (%) of the best solutions for $n = 100$. The results of GLS with $b = 1$, where no mutation is added, and the results of MLS are also included for comparison purposes.

In all cases, CX1 exhibited poor performance. One of the conceivable reasons of this phenomena is that CX1 can generate only children with limited variety. On the whole, we can summarize these results as follows. (1) GLS can obtain solutions of higher quality than MLS, provided that rather long computational time is allowed. (2) GLS is rather insensitive to the type of perturbations to generate new solutions, crossover and/or mutation. If only one of them is required, crossover appears slightly more effective than mutation. (3) The solution quality critically depends on the type

of neighborhood. (4) The convergence of GLS becomes faster by bounding the search length of LS (e.g., $b = 1$); however, the solution quality becomes slightly more sensitive to crossover than GLS with $b = \infty$.

5. Comparison and Conclusion

We conclude this paper by comparing four metaheuristic algorithms tested so far. Figures 1 and 2 show how the average error (%) of the best solutions improves as the number of samples increases. Both neighborhoods N_{ins} (Fig. 1) and N_{swap} (Fig. 2) are examined. In the case of GA, mutation operators INS1 and SWAP1 are used corresponding to N_{ins} and N_{swap}, respectively. For GRASP with neighborhood N_{ins} (resp., N_{swap}), local gain function e_6 (b) (resp., e_6 (f)) is used and parameter $|CA|$ is set to 7 (resp., 4). Selection strategy BESTP and crossover operator OXU are used for GA and GLS. The parameter $|P|$ is set to 1000 for GA and 20 for GLS, and mutation is not incorporated for GLS.

From these results, we can conclude that: (1) Performance of GA is robust about mutation; however, its performance is rather poor. (2) GRASP and GLS further improve the performance of MLS. GLS is more powerful than GRASP, if the same neighborhood is used. (Recall that the performance of GLS is quite robust about crossover, while the performance of GRASP is very sensitive to the local gain functions used to generate initial solutions.) (3) Performance of MLS, GRASP and GLS is affected by the type of neighborhood.

In view of these, we can summarize our recommendation as follows.

1. If the simplicity is our first concern, use MLS. In this case, the component to be defined in correspondence with the given problem is only the neighborhood.
2. If obtaining solutions of higher quality is important, use GLS. The simplest form of GLS is to introduce only mutations, in order to enhance MLS. As mutation can be realized by using the neighborhood of LS, the additional efforts required is very little, as far as the problem oriented efforts are concerned. It would also be worthwhile to introduce crossovers as well, since it may improve the performance further.

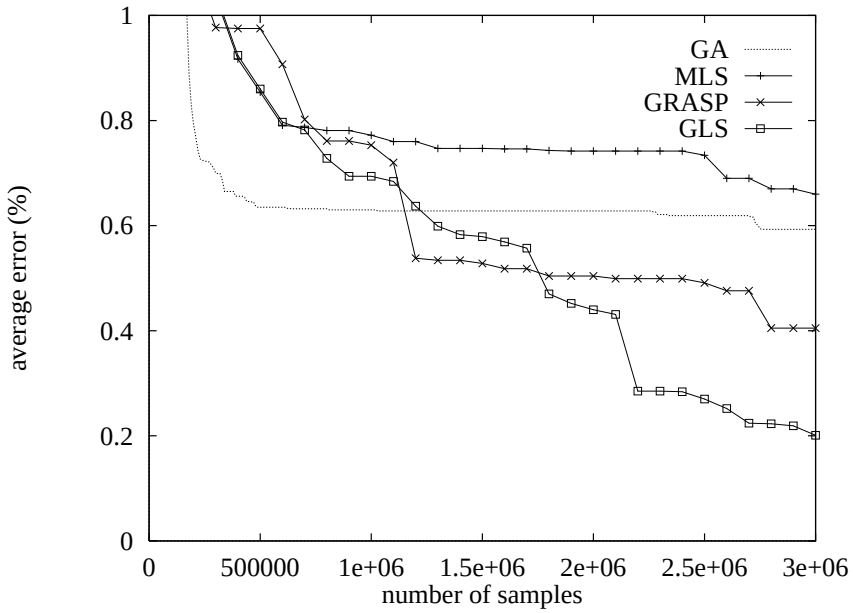

Figure 1: Average error (%) of the best solutions against the number of samples (neighborhood N_{ins} and mutation INS1 are used).

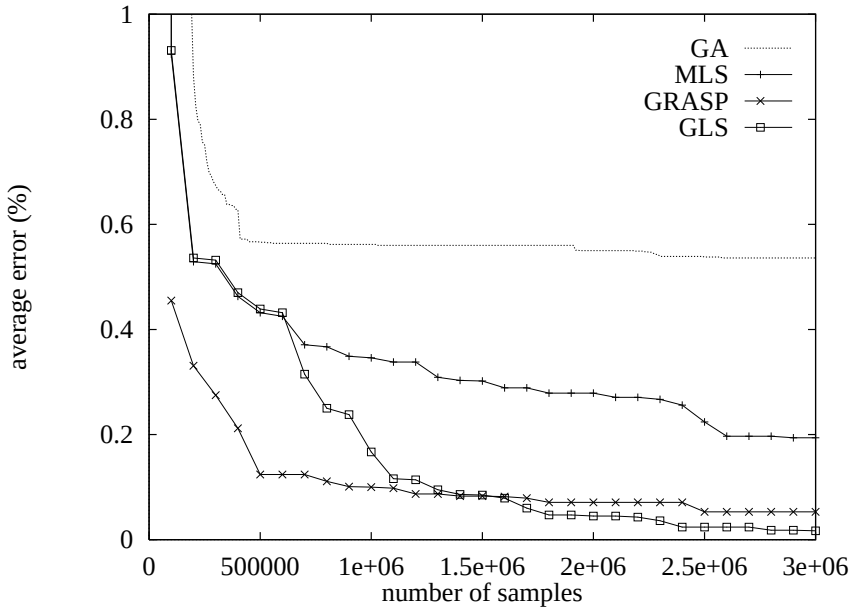

Figure 2: Average error (%) of the best solutions against the number of samples (neighborhood N_{swap} and mutation SWAP1 are used).

6. References

[1] K.D. Boese, A.B. Kahng and S. Muddu, A New Adaptive Multi-Start Technique for Combinatorial Global Optimizations, *Operations Research Letters*, 16 (1994) 101.

[2] L. Davis, Applying Adaptive Algorithms to Epistatic Domains, in: *Proceedings of the 9th IJCAI*, ed. A. Joshi (Morgan Kaufmann, California, 1985) p. 162.

[3] L. Davis, *Handbook of Genetic Algorithms*, (Van Nostrand Reinhold, New York, 1991).

[4] T.A. Feo, K. Venkatraman and J.F. Bard, A GRASP for a Difficult Single Machine Scheduling Problem, *Computers and Operations Research*, 18 (1991) 635.

[5] T.A. Feo, M.G.C. Resende and S.H. Smith, A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set, *Operations Research*, 42 (1994) 860.

[6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, (Freeman, New York, 1979).

[7] D.E. Goldberg and R. Lingle, Alleles, Loci, and the Traveling Salesman Problem, in: *Proceedings of the 1st ICGA*, ed. J.J. Grefenstette (Lawrence Erlbaum Associates, New Jersey, 1985) p. 154.

[8] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, (Addison-Wesley, Massachusetts, 1989).

[9] J. Grefenstette, R. Gopal, B. Rosmaita and D. Van Gucht, Genetic Algorithms for the Traveling Salesman Problem, in: *Proceedings of the 1st ICGA*, ed. J.J. Grefenstette (Lawrence Erlbaum Associates, New Jersey, 1985) p. 160.

[10] J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, (MIT Press, Massachusetts, 1992).

[11] A. Homaifar, S. Guan and G.E. Liepins, A New Approach on the Traveling Salesman Problem by Genetic Algorithms, in: *Proceedings of the 5th ICGA*, ed. S. Forrest (Morgan Kaufmann, California, 1993) p. 460.

[12] T. Ibaraki and Y. Nakamura, A Dynamic Programming Method for Single Machine Scheduling, *European Journal of Operational Research*, 76 (1994) 72.

[13] A. Kolen and E. Pesch, Genetic Local Search in Combinatorial Optimization, *Discrete Applied Mathematics*, 48 (1994) 273.

[14] M. Laguna, T.A. Feo and H.C. Elrod, A Greedy Randomized Adap-

tive Search Procedure for the Two-Partition Problem, *Operations Research*, 42 (1994) 677.

[15] S. Lin and B.W. Kernighan, An Effective Heuristic Algorithm for the Traveling Salesman Problem, *Operations Research*, 21 (1973) 498.

[16] H. Mühlenbein, M. Gorges-Schleuter and O. Krämer, Evolution Algorithms in Combinatorial Optimization, *Parallel Computing*, 7 (1988) 65.

[17] I.M. Oliver, D.J. Smith and J.R.C. Holland, A Study of Permutation Crossover Operators on the Traveling Salesman Problem, in: *Proceedings of the 2nd ICGA*, ed. J.J. Grefenstette (Lawrence Erlbaum Associates, New Jersey, 1987) p. 224.

[18] C.N. Potts and L.N. Van Wassenhove, A Decomposition Algorithm for the Single Machine Total Tardiness Problem, *Operations Research Letters*, 1 (1982) 177.

[19] C.N. Potts and L.N. Van Wassenhove, A Branch and Bound Algorithm for the Total Weighted Tardiness Problem, *Operations Research*, 33 (1985) 363.

[20] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley and C. Whitley, A Comparison of Genetic Sequencing Operators, in: *Proceedings of the 4th ICGA*, eds. R.K. Belew and L.B. Booker (Morgan Kaufmann, California, 1991) p. 69.

[21] G. Syswerda, Simulated Crossover in Genetic Algorithms, in: *Proceedings of the 2nd FOGA*, ed. L.D. Whitley (Morgan Kaufmann, California, 1993) p. 239.

[22] N.L.J. Ulder, E.H.L. Aarts, H.-J. Bandelt, P.J.M. Van Laarhouven and E. Pesch, Genetic Local Search Algorithms for the Traveling Salesman Problem, in: *Proceedings of the 1st PPSN*, eds. H.-P. Schwefel and R. Männer (Springer-Verlag, Berlin, 1990) p. 109.

[23] D. Whitley, T. Starkweather and D. Fuquay, Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator, in: *Proceedings of the 3rd ICGA*, ed. J.D. Schaffer (Morgan Kaufmann, California, 1989) p. 133.

[24] M. Yagiura and T. Ibaraki, On Genetic Crossover Operators for Sequencing Problems (in Japanese), *T. IEE Japan*, 114-C (1994) 713.