Metaheuristics as Robust and Simple Optimization Tools

Mutsunori Yagiura
Graduate School of Engineering
Kyoto University
Kyoto 606-01, Japan
yagiura@kuamp.kyoto-u.ac.jp

Toshihide Ibaraki
Graduate School of Engineering
Kyoto University
Kyoto 606-01, Japan
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 metaheuristics, such as random multi-start local search (MLS), genetic algorithm (GA), simulated annealing (SA) and tabu search (TS), using rather simple inside operators. The results indicate that: (1) simple implementation of MLS is usually competitive with (or even better than) GA, (2) GA combined with local search is quite effective if longer computational time is allowed, and its performance is not sensitive to crossovers, (3) SA is also quite effective if longer computational time is allowed, and its performance is not much dependent on parameter values, (4) there are cases in which TS is more effective than MLS; however, its performance depends on how to define the tabu list and parameter values and (5) the definition of neighborhood is very important for all of MLS, SA and TS.**

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

I. Introduction

There are numerous combinatorial optimization problems, for which computing exact optimal solutions is computationally intractable, e.g., those known as *NP-hard*. However, in practice, we are usually satisfied with good solutions, and in this sense, *approximate* (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) [3; 13], *genetic algorithm* (abbreviated as GA) [7; 8], *simulated annealing* (abbreviated as SA) [10; 11], *tabu search* (abbreviated as TS) [5; 6] 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 exhibit reasonably good performance, much better than those obtainable by simple heuristics. We pursue this direction more carefully in this paper, by implementing various metaheuristics 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.

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.

II. Metaheuristics

In this section, we describe general frameworks of various metaheuristic algorithms.

Among basic strategies of approximate algorithms are

- greedy method
- local search (LS)

and so on. 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 $N(x)$, where $N(x)$ is a set of solutions obtained by slightly perturbing x. A solution x is called *locally optimal*, if no better solution exists in $N(x)$. We call the computation of obtaining a locally optimal solution from an initial x as a *trial* of LS.

In general, if only one trial of LS is applied, solutions with much better quality may remain unvisited. Therefore, LS may be enhanced by:

- trying many initial solutions,
- using a sophisticated neighborhood or a larger neighborhood,
- using a sophisticated search strategy, sometimes allowing moves to worse solutions in $N(x)$.

Metaheuristics such as MLS, GA, SA and TS can be viewed as such variants of LS.

MLS: LS is repeated from a number of initial solutions and the best solution found during the entire search is output. The initial solution may be generated randomly or by using greedy methods. The greedy randomized adaptive search procedure (GRASP) [3; 4] is a variant of MLS, in which the initial solutions are generated by randomized greedy methods. Another variant of MLS called iterated local search (abbreviated as ILS) [9; 14] is also possible, where the initial solutions are chosen from the neighborhood $N(x)$ of the locally optimal solution x found in the previous trial.

GA: The genetic algorithm is a probabilistic algorithm, whose idea comes from evolution, 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 $N(P)$ is defined to be the set of solutions obtainable from P by

crossover and mutation operators. A crossover operator generates one or more solutions (*children*) by combining two or more candidate solutions (*parents*), and a mutation operator generates a solution by slightly perturbing a candidate solution. The GA starts from an initial candidate solutions P and repeat replacing P with $P' \subset P \cup N(P)$ according to the selection rule. A variant of GA in which solutions generated by the crossover and mutation operators are improved by LS is called the *genetic local search* (abbreviated as GLS) [1; 12; 15; 16]. 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.

SA: This is a variant of LS, in which test solutions are randomly chosen from $N(x)$ and accepted with probability that is 1 if the test solution is better than x, and positive even if the test solution is worse than x. By giving a positive probability to move to a worse solution, the search is able to escape from poor locally optimal solutions. The acceptance probability is judiciously controlled by a parameter called *temperature*, whose idea stems from the physical annealing process.

TS: The tabu search tries to enhance LS by using memory of the previous search. Basically the best solution in $N(x)\backslash(\{x\} \cup T)$ is chosen as the next solution, where the set T, called *tabu list* (or *short term memory*), is a set of solutions which includes those solutions most recently visited. Within this restricted neighborhood, a move to a new solution is always executed even if the current solution is already locally optimal, but cycling of a short period can be avoided as a result of introducing T. Another type of memory, called *long term memory*, is often employed in the framework of TS, which memorizes the past search information such as the frequency that a decision variable has been changed, the frequency that a solution has been visited and so on. This memory is used to direct the search to the unvisited region (i.e., *diversification*).

III. 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 $\sigma(k) = i$ means that i 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 $c_i = \sum_{k=1}^{\sigma^{-1}(i)} p_{\sigma(k)}$. A sequence σ is optimal if it minimizes

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

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.

IV. Computational Results

Some details of the tested algorithms and the computational results are discussed in this section. All the tested algorithms were coded in C language and run on a Sun SPARC station IPX. The quality of the obtained solutions is evaluated by the average error from the best cost values, which were found in the entire experiment. The efficiency of algorithms is measured on the basis of the number of the solution samples evaluated, rather than the computational time, since the computational time depends on the computers used and other factors such as programming skill. Ten problem instances for each of $n = 35$ and 100 are generated as described in [17].

Initial solutions are generated randomly except for GRASP, and the neighborhood $N(\sigma)$ is always scanned according to a prespecified random order.

A. Random Multi-Start Local Search

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

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

(2) Search strategies: FIRST scans $N(\sigma)$ and selects the first improved solution σ' satisfying $cost(\sigma') < cost(\sigma)$, and BEST selects the solution σ' having the best cost in the entire area of $N(\sigma)$.

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 were generated for each test run with $n = 35$ and 100, respectively. Table 1 also shows the average number of trials (i.e., the number of initial solutions) in parentheses.

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

	n = 35; 3 X 10^5 samples		n = 100; 3 X 10^6 samples	
	N_{ins}	N_{swap}	N_{ins}	N_{swap}
FIRST	0.000 (97.6)	0.000 (131.0)	0.667 (103.6)	0.180 (126.5)
BEST	0.289 (8.6)	0.047 (13.9)	4.694 (3.2)	0.622 (5.0)

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 remaining experiments.) (2) The quality of solutions obtained by neighborhood N_{swap} is better than that obtained by N_{ins}.

B. Greedy Randomized Adaptive Search Procedure

The initial solutions are generated as follows. At each step, let $M = \{i \in V \mid i \text{ is not scheduled yet}\}$. A candidate set $CA \subseteq M$ of a fixed number of jobs ($|CA|$ is a prespecified positive integer) is chosen according to a criterion based on a local gain function that represents greedy heuristics, and then a job $i \in CA$ is randomly chosen as

the next job.

A total of 12 local gain functions e_i(f) and e_i(b) defined in [17] 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. The details of the local gain functions are omitted here because of lack of space. In this subsection, only the results with the neighborhood N_{swap} are shown; however, similar tendencies were observed for N_{ins}. Table 2 shows the average

Table 2. 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	0.052	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				0.180			314.6

error (%) of the best solutions obtained by various GRASP algorithms within 3×10^6 samples. For 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 initial solutions generated by greedy methods, i.e., with $|CA| = 1$, and the bottom is the average error of the initial solutions generated randomly.

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

C. Iterated Local Search

When a locally optimal solution σ is reached, the next initial solution is generated by applying one uphill move, which is either the best solution in $N(\sigma)$ or randomly chosen from $N(\sigma)$. A counter UC and a parameter R are used to control this selection process. If $UC < R$ holds, then the best solution in $N(\sigma)$ is chosen as the next initial solution and UC is increased by one; otherwise the next solution is randomly chosen from $N(\sigma)$ and UC is reset to zero. Finally, if a solution better than the past best solution is found, UC is reset to zero.

Here we tested the parameter values $R = 0, 1, 2, 4, 8$. Table 3 shows the average error (%) of the best solutions, where 3×10^5 (resp., 3×10^6) samples were generated for $n = 35$ (resp., 100).

The results indicate that: (1) ILS is quite effective if N_{swap} is used. Its performance is robust about the parameter R, however, (2) ILS does not improve the performance of MLS if N_{ins} is used. In conclusion, ILS is simple (even simpler than GRASP) and can be more powerful than MLS if proper neighborhood is employed.

Table 3. Average error in % of the best solutions with ILS.

	$n = 35$; 3×10^5 samples		$n = 100$; 3×10^6 samples	
R	N_{ins}	N_{swap}	N_{ins}	N_{swap}
0	0.000	0.000	1.145	0.078
1	0.000	0.000	1.702	0.057
2	0.044	0.000	2.338	0.111
4	0.285	0.000	2.584	0.114
8	0.901	0.036	2.664	0.151
MLS	0.000	0.000	0.667	0.180

D. Genetic Algorithm

The framework of GA we examined is as follows: at each generation, generate a set of solutions $Q \subseteq N(P)$ and select a set of $|P|$ solutions P' from $P \cup Q$, and set $P := P'$. Recall that $N(P)$ is the set of solutions obtainable from P by crossover and mutation operators. Among various types of crossover, mutation and selection operators, we considered the following representative operators.

Crossover: The *order crossover* [2; 15] is employed here. We assume that one child σ_C is produced from two parents σ_A and σ_B. Generate randomly an n-bit mask $m \in \{0,1\}^n$. Set $\sigma_C(k) := \sigma_A(k)$ for all k satisfying $m(k) = 0$. 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 $S_m = \{\sigma_A(k) \mid m(k) = 0\}$. 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 k satisfying $m(k) = 1$ according to D'_B. Crossover operators based on arbitrary masks are called uniform, and those based on restricted masks having at most r adjacent 0-1 pairs are called r-point; e.g., masks 11000 and 10011 are 1-point and 2-point respectively. We call 1-point, 2-point and uniform crossover operators of this type as OX1, OX2 and OXU respectively.

Other types of crossover operators, such as partially mapped crossover and cycle crossover [17], were also examined; however, the results are omitted here because of the lack of space.

Mutation: Mutation employed in this paper perturbs a candidate solution σ by a random selection $\sigma' \in N(\sigma)$ and $\sigma := \sigma'$. Two types of neighborhood $N_{ins}(\sigma)$ and $N_{swap}(\sigma)$ are used as $N(\sigma)$; the resulting mutations are denoted as INS and SWAP respectively.

Selection: In our experiment, the set of generated candidate solutions $Q \subseteq N(P)$ is determined as follows. We use $|Q| = 1$ and the child $\sigma_C \in Q$ is obtained by randomly selecting two parents σ_A, $\sigma_B \in P$, mutating either σ_A or σ_B, and then applying crossover of σ_A and σ_B. Then with a solution σ_{worst} satisfying $cost(\sigma_{worst}) \geq cost(\sigma)$ for all $\sigma \in P \cup Q$, let $P' := P \cup \{\sigma_C\} - \{\sigma_{worst}\}$. Note that the selection is executed only if the child $\sigma_C \in Q$ is not in P.

Various GA defined by the above crossover, mutation and selection operators are compared, in which $|P|$ is always set to 100. Note that exactly one sample (i.e., cost evaluation) occurs during one generation, since $|Q| = 1$ is used. Table 4 shows the average error (%) of the best so-

lutions obtained by the tested algorithms, where 3×10^4 (resp., 3×10^5) samples were allowed for $n = 35$ (resp., 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.

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	INS	SWAP	no MUT	INS	SWAP
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	4.0	—	1.4	1.5
MLS	—	0.1	0.06	—	1.0	0.5

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) 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. (4) MLS performs better than GA of this sort.

GA using different population sizes $|P|$ are also tested. Table 5 shows the average error in % of the best solutions after 3×10^6 samples were generated, where $n = 100$ and crossover type is OXU. For comparison purpose, the results of MLS are also included in Table 5, where the same neighborhood as the mutation is used.

Table 5. Average error in % of the best solutions with GA after 3×10^6 samples.

$\|P\|$	10^1	10^2	10^3	10^4	MLS
INS	4.874	0.639	0.593	0.522	0.667
SWAP	0.802	0.800	0.536	0.325	0.180

The results indicate that: (1) Better solutions are obtained on 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.

E. Genetic Local Search

Genetic local search (GLS) is a variation of GA, in which the new candidate solutions in Q are improved by LS. Other operators are the same as GA. Various GLS were compared, in which $|P|$ is set to 20. The results for $n = 35$ are omitted, since almost all the tested algorithms could obtain exact optimal solutions for all the instances. Table 6 shows the average error (%) of the best solutions for $n = 100$, where 3×10^6 samples were generated. The results of MLS are also included for comparison purposes.

We can summarize these results as follows. (1) GLS can obtain solutions of higher quality than GA and MLS, particularly when long computational time is allowed. (2)

Table 6. Average error (%) of the best solutions with GLS in which $|P| = 20$.

neighbor	N_{ins}			N_{swap}		
mutation	noMUT	INS	SWAP	noMUT	INS	SWAP
OX1	0.279	0.359	0.301	0.088	0.062	0.067
OX2	0.218	0.325	0.270	0.036	0.049	0.045
OXU	0.201	0.162	0.264	0.013	0.067	0.036
MUT only	—	0.596	0.508	—	0.100	0.054
MLS		0.667			0.180	

GLS is rather insensitive to the types of crossover and mutation. Between crossover and mutation, crossover appears slightly more effective. On the other hand, GLS only with mutations is much easier to implement, since mutation can be realized by using the neighborhood of LS, and hence the additional efforts required is very little. (3) The solution quality critically depends on the type of neighborhood.

ILS can be viewed as a special case of GLS in which $|P| = 1$ and crossover is not incorporated. By comparing Tables 3 ($R = 0$) and 6 (column 'MUT only'), we can conclude that GLS improves the performance of ILS.

F. Simulated Annealing

The SA used in this paper is similar to the one in [10]. Our algorithm includes parameters IP, TR, SF and TF. The initial temperature t is determined so that $(\sum_{\sigma' \in UP(\sigma)} e^{-\{cost(\sigma') - cost(\sigma)\}/t})/|UP(\sigma)| \simeq IP$ for randomly chosen initial solutions σ, where $UP(\sigma) = \{\sigma' \in N(\sigma) \mid cost(\sigma') > cost(\sigma)\}$. Then the following loop is executed, where k is initially set to 0.

While $k < TR \cdot |N|$ do the following.

(a) Perform the following loop $SF \cdot |N|$ times.

i. Pick a random neighbor $\sigma' \in N(\sigma)$.

ii. Let $\Delta = cost(\sigma') - cost(\sigma)$.

iii. If $\Delta \leq 0$, set $\sigma = \sigma'$.

iv. If $\Delta > 0$, set $\sigma = \sigma'$ with probability $e^{-\Delta/t}$.

v. If $\Delta < 0$, set $k = 0$; otherwise set $k = k + 1$.

(b) Set $t = TF \cdot t$.

Upon termination, the search is restarted from a randomly chosen initial solution unless sufficient number of solution samples has been tested.

The parameter TR is set to 1 for N_{ins} and 2 for N_{swap} according to a preliminary experiment. The parameter TF is fixed to 0.95 as suggested in [10].

First the effect of IP is examined, and it is observed that: (1) the quality of the obtained solutions becomes better as IP increases up to 0.1; however, it does not change much if $IP \geq 0.1$, and (2) the number of samples needed for one trial of the algorithm becomes larger as IP increases.

Next the effect of SF is examined with $IP = 0.3$. It is observed that: (1) the quality of the solutions becomes better as SF increases up to 1; however, it does not change much if $SF \geq 1$, and (2) the number of samples needed for one trial of the algorithm becomes larger as SF increases.

From these results, we examined four combinations of IP and SF: $IP = 0.1, 0.3$ and $SF = 1, 2$. Table 7 shows the average error (%) of the best solutions, where 3×10^5 (resp., 3×10^6) samples were allowed for $n = 35$ (resp., $n = 100$). The results of MLS are also included for comparison.

Table 7. Average error (%) of the best solutions with SA.

		$n = 35; 3 \times 10^5$ samples		$n = 100; 3 \times 10^6$ samples	
IP	SF	N_{ins}	N_{swap}	N_{ins}	N_{swap}
0.1	1	0.144	0.000	0.183	0.032
0.1	2	0.124	0.003	0.215	0.030
0.3	1	0.035	0.000	0.151	0.027
0.3	2	0.033	0.000	0.225	0.012
MLS		0.000	0.000	0.667	0.180

We can summarize these results as follows. (1) SA can obtain solutions of higher quality than MLS, provided that rather long computational time is allowed. (2) The quality of the solutions obtained by SA is rather insensitive to the parameter values, though the number of samples needed for one trial critically depends on them. (3) The solution quality critically depends on the type of neighborhood.

G. Tabu Search

In the tabu search, we scan $N(\sigma)$ and select the first solution σ' satisfying $cost(\sigma') < cost(\sigma)$ and $\sigma' \notin T \cup \{\sigma\}$, or satisfying $cost(\sigma') < bestcost$ (i.e., *aspiration criterion*), where *bestcost* is the cost of the best solution found duing the past search. If none of the solutions in $N(\sigma)$ is selected by the above rule (i.e., σ is locally optimal), the next solution is selected as follows. A counter MC and a parameter R are used to control the process of generating the next initial solution when one trial of tabu search ends. If $MC < R$ holds, then the best solution in $N(\sigma)\backslash(\{\sigma\} \cup T)$ is chosen as the next initial solution, and MC is incremented by one; otherwise the solution $\sigma' \in N(\sigma)\backslash(\{\sigma\} \cup T)$ that minimizes $cost(\sigma') + \alpha \cdot penalty(\sigma')$ is chosen as the next solution and MC is reset to zero, where α is a prespecified program parameter and $penalty(\sigma')$ is a cost of the long term memory. Finally, if a solution better than the past best solution is found (aspiration criterion), MC is reset to zero.

Two types of tabu lists T_{job} and T_{pos} are considered, where $T_{job}(\sigma) = \{\sigma' \in N(\sigma) \mid$ the move from σ to σ' changes the position of a job whose position has been changed in the last TT moves$\}$ and $T_{pos}(\sigma) = \{\sigma' \in N(\sigma) \mid$ the move from σ to σ' assigns a job to the position where it has been assigned in the last TT moves$\}$. The parameter TT is a prespecified nonnegative integer called *tabu tenure*. We examined two types of penalties, which are called $penalty_{move}$ and $penalty_{period}$. Let $LT_{move}(i,k)$ be the number of moves of job i from position k which have been made during the past search, and $LT_{period}(i,k)$ be the period that job i has been scheduled at position k so far. Then $penalty_{move}(\sigma') = \sum_{j \in CH(\sigma')} LT_{move}(j, \sigma^{-1}(j))$ and $penalty_{period}(\sigma') = \sum_{k=1}^{n} LT_{period}(\sigma'(k), k)$, where $CH(\sigma')$ is the set of jobs whose positions are changed by the move from σ to σ'.

First α is set to 0 in order to examine the effect of short term memory. Two types of tabu list T_{job} and T_{pos}, and various TT values are tested. It is observed that: (1) TS can obtain solutions of higher quality than MLS if N_{swap} and T_{pos} are used and $TT = 1 \sim 5$, and (2) the performance of TS is worse than MLS with other combinations. From these, we consider only T_{pos} in the remaining experiments.

Second the effect of the long term memory is examined. Two types of penalties $penalty_{move}$ and $penalty_{period}$, and various α values are tested. It is observed that: (1) TS can obtain solutions of better quality than MLS if N_{swap} and $penalty_{period}$ are used with $\alpha \geq 10^4$, and (2) the performance of TS is worse than MLS with other combinations. From these, we consider only $penalty_{period}$ in the remaining experiments.

Finally the short and long term memories are combined together. Here the parameter α is set to 10^3 (resp., 10^4) for $n = 35$ (resp., 100). Parameter values $TT = 1, 3, 5, 7, 10, 20$ are examined and R is set to TT. Table 8 shows the average error (%) of the best solutions obtained by TS, where 3×10^5 (resp., 3×10^6) samples were generated for $n = 35$ (resp., 100).

Table 8. Average error (%) from the best solutions with TS.

	$n = 35; 3 \times 10^5$ samples		$n = 100; 3 \times 10^6$ samples	
TT	N_{ins}	N_{swap}	N_{ins}	N_{swap}
1	0.000	0.000	1.447	0.084
3	0.037	0.000	0.963	0.148
5	0.084	0.000	1.645	0.091
7	0.358	0.000	1.687	0.165
10	0.360	0.000	1.682	0.146
20	0.542	0.000	1.897	0.202
MLS	0.000	0.000	0.667	0.180

We can summarize these results as follows. (1) TS can obtain solutions of higher quality than MLS if N_{swap} is used, and its performance is not sensitive to parameter TT. (2) TS does not improve the performance of MLS if N_{ins} is used. (3) The performance of TS critically depends on the type of memories (the tabu list and the long term memory).

V. 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 for problem instances with $n = 100$. Both neighborhoods N_{ins} (Fig. 1) and N_{swap} (Fig. 2) are examined, where in the case of GA, mutation operators INS and SWAP 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). For ILS with neighborhood N_{ins} (resp., N_{swap}), parameter R is set to 0 (resp., 1). Crossover operator OXU is 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. For SA with neighborhood N_{ins} (resp., N_{swap}), parameters are set to $IP = 0.3$, $TR = 1$, $SF = 1$ and $TF = 0.95$ (resp., $IP = 0.3$, $TR = 2$, $SF = 2$ and $TF = 0.95$). For TS with neighborhood N_{ins} (resp., N_{swap}), tabu list T_{pos} and penalty $penalty_{period}$ are used and the parameters are set to $R = 3$, $TT = 3$ and $\alpha = 10^4$ (resp., $R = 5$, $TT = 5$ and $\alpha = 10^4$).

From these results, we can conclude that: (1) Performance of GA is robust about mutation; however, its performance is rather poor. (2) GRASP, GLS and SA im-

prove the performance of MLS further, among which SA and GLS appear more powerful if the same neighborhood is used. (3) Performance of MLS, GRASP, ILS, GLS, SA and TS critically depends on 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 is only the neighborhood.
2. If obtaining solutions of higher quality is important, use SA or GLS. Detailed parameter tuning is not important for SA and GLS provided that sufficient amount of computational time is available.

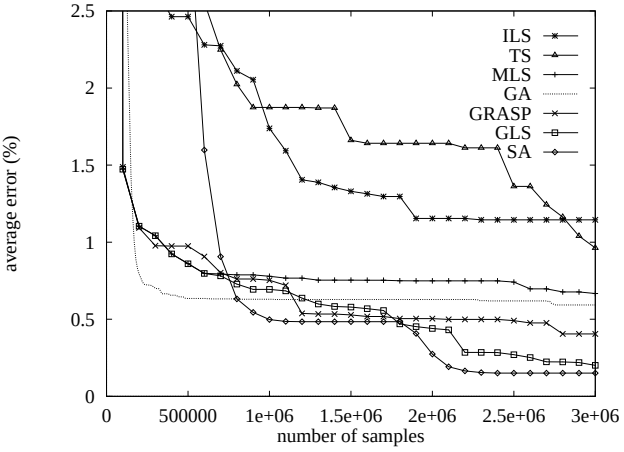

Fig. 1. Average error (%) from the best solution (N_{ins}).

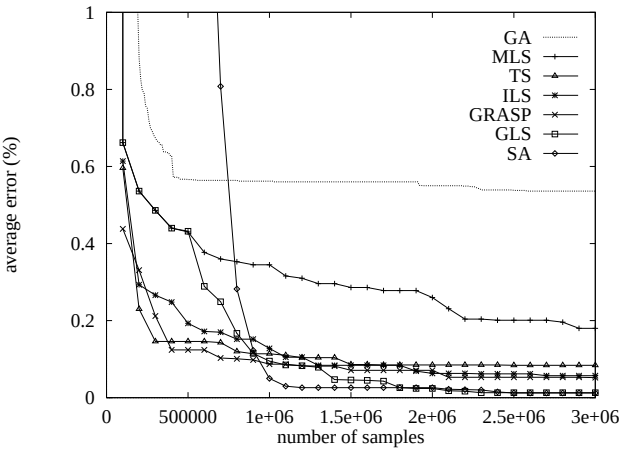

Fig. 2. Average error (%) from the best solution (N_{swap}).

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) 101–113.

[2] L. Davis, "Applying Adaptive Algorithms to Epistatic Domains," in: *Proc. 9th IJCAI*, (Morgan Kaufmann, 1985) p. 162–164.

[3] T.A. Feo, K. Venkatraman and J.F. Bard, "A GRASP for a Difficult Single Machine Scheduling Problem," *Computers and Operations Research*, 18 (1991) 635–643.

[4] 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) 860–878.

[5] F. Glover, "Tabu Search — Part I," *ORSA J. Computing*, 1 (1989) 190–206; Part II, ditto, 2 (1990) 4–32.

[6] F. Glover, E. Taillard and D. de Werra, "A User's Guide to Tabu Search," *Annals of Operations Research*, 41 (1993) 3–28.

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

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

[9] D.S. Johnson, "Local Optimization and the Traveling Salesman Problem," in: *Proc. 17th Colloquium on Automata, Languages and Programming*, (Springer-Verlag, 1990) p. 446–461.

[10] D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon, "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning," *Operations Research*, 37 (1989) 865–892; "Part II, Graph Coloring and Number Partitioning," ditto, 39 (1991) 378–406.

[11] S. Kirkpatrick, C.D. Gelatt, Jr. and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, 220 (1983) 671–680.

[12] A. Kolen and E. Pesch, "Genetic Local Search in Combinatorial Optimization," *Discrete Applied Mathematics*, 48 (1994) 273–284.

[13] S. Lin and B.W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, 21 (1973) 498–516.

[14] O. Martin, S.W. Otto and E.W. Felten, "Large-Step Markov Chains for the TSP Incorporating Local Search Heuristic," *Operations Research Letters*, 11 (1992) 219–224.

[15] H. Mühlenbein, M. Gorges-Schleuter and O. Krämer, "Evolution Algorithms in Combinatorial Optimization," *Parallel Computing*, 7 (1988) 65–85.

[16] 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: *Proc. 1st PPSN*, (Springer-Verlag, 1990) p. 109–116.

[17] M. Yagiura and T. Ibaraki, "Genetic and Local Search Algorithms as Robust and Simple Optimization Tools," in: *Proc. 1st Metaheuristics International Conference* (Kluwer, to appear).