Experimental Analyses of the Life Span Method for the Quadratic Assignment Problem

Katsuki FUJISAWA
Department of Mathematical and Computing Sciences,
Tokyo Institute of Technology,
Oh-Okayama, Meguro-ku, Tokyo, Japan
Mikio KUBO
Department of Information Engineering and Logistics,
Tokyo University of Mercantile Marine,
Etsujima, Koutou-ku, Tokyo, Japan

Abstract

In this paper, we report an application of the life span method (LSM), a variant of tabu search introduced by the authors, to the quadratic assignment problem which has applications on facility location and backboard wiring, etc. We discuss how to adapt the LSM to the quadratic assignment problem and compare the performance with previous heuristics. The main purpose of this paper is to perform experimental analyses composed of optimizing the various parameters and to estimate the performance not only in the best case but the average behavior.
Key words: life span method, tabu search, combinatorial optimization, approximate algorithms, experimental analysis, quadratic assignment problem.

1 Introduction

The Quadratic Assignment Problem (QAP) is a combinatorial optimization problem having many applications including facility location, ordering of data on a disk, backboard wiring, machine scheduling, analyzing chemical, the location of departments (or offices), etc. [9].

In the context of facility location, a set of n facilities is to be assigned to an equal number of locations. The QAP is defined as follows:

Definition 1 (Quadratic Assignment Problem: QAP)
Given a set $V = \{1, \cdots, n\}$ and $n \times n$ symmetric matrices $F = (f_{ij})$ and $D = (d_{k\ell})$, find a permutation $\pi : V \to \{1, \cdots, n\}$ which minimizes the cost function

$$c(\pi) = \sum_i \sum_j f_{ij} d_{\pi(i)\pi(j)}.$$

This problem has the following interpretation. A permutation π is the assignment of n objects (facilities) to n locations. The value f_{ij} is the flow between objects i and j, and $d_{k\ell}$ is the distance between locations k and ℓ. When object i is assigned to location k and object j is assigned to location ℓ, the cumulative distance $f_{ij}d_{k\ell}$ is incurred. The objective of the QAP is to find an assignment which minimizes the cumulative distances between all pairs of objects.

The QAP is known to be one of the hardest problems in $\mathcal{NP}$-hard problems because there are many local optimal solutions that are very near to the global optima.

Using the 0-1 variable x_{ij} which is set to 1 if object i is assigned to location j, the QAP is stated as the following quadratic integer programming problem:

$$\min \sum_i \sum_j \sum_k \sum_\ell f_{ij} d_{k\ell} x_{ik} x_{j\ell} \tag{1}$$

subject to

$$\sum_j x_{ij} = 1 \quad i \in V, \tag{2}$$

$$\sum_j x_{ji} = 1 \quad i \in V, \tag{3}$$

$$x_{ij} \in \{0,1\} \quad i \in V, j \in V. \tag{4}$$

The set of feasible solutions corresponds to the set of $n \times n$ permutation matrices (x_{ij}), where x_{ij} is equaled to 1 if $\pi(i) = k$. Many classical combinatorial problems, such as the traveling salesman problem and the graph partitioning problem, are special cases of the QAP.

In this paper, we estimate the performance of the life span method (LSM) with extensive numerical experiments which have been carried out on the QAP.

The organization of this paper is as follows: In Section 2, we describe the pervious work for the QAP. In Section 3, we briefly review the LSM, a variant tabu search introduced by the authors [21]. In Section 4, we give an application of the LSM to the QAP. The results of the numerical experiments and parameter optimization are shown in Section 5. The final section gives conclusions.

2 Pervious Work

In this section, we briefly review the previous work on exact and approximation approaches to the QAP, and the benchmark problems for the QAP.

2.1 Exact algorithms

Because the QAP is especially intractable in $\mathcal{NP}$-hard problems, exact solution approaches have been limited to small ($n \leq 20$) problems, and most of them are based on the branch and bound method. The lower bounds for the QAP tend to deteriorate quickly, as the size of the QAP increases.

The lower bound algorithms for the QAP can be categorized into four groups. The first group of bounds is the Gilmore-Lawler lower bound [12, 22] and related bounds. Eigenvalue-based approaches [9] constitute the second category. The third group of bounds is mainly based on reformulations of the QAP [8, 11, 20]. The forth group is based on the semidefinite relaxations [30].

2.2 Approximate algorithms

Several metastrategies have been applied to find approximate solutions to the QAP.

Wilhelm and Ward [31], Connolly [5] and Burkard and Rendl [3] proposed several variations of simulated annealing for the QAP. Taillard [28] applied tabu search to the QAP. He used the dynamic tabu list of varying tabu list size randomly in the problem-dependent interval and developed a parallel implementation on a ring 10 transputers. Skorin-Kapov [26, 27] also applied tabu search to the QAP. Chakrapani and Skorin-Kapov [6] compared tabu search with simulated annealing and neural net approaches. Chakrapani and Skorin-Kapov [7] developed a massively parallel implementation of tabu search, called *Augmented Par_tabu* on the Connection Machine CM-2. Their implementation used n^2 processors. Their algorithm includes dynamic tabu list, aspiration criterion, and long term memory to escape a local optimum. Fleurent and Ferland [10] proposed the genetic hybrid algorithm. Li et al. [23] proposed a greedy randomized adaptive search procedure (GRASP) for the QAP.

2.3 QAPLIB – A Quadratic Assignment Problem Library

QAPLIB (a Quadratic Assignment Problem LIBrary) is a collection of problem instances for the QAP of size $n \geq 8$ [2]. Some of these problems are generated by various researchers for their own testing purposes. QAPLIB is composed of the test problems, the best known or optimal solutions, and the best upper and lower bounds . QAPLIB is available via anonymous ftp at **ftp.tu-graz.ac.at** in the directory /pub/papers/qaplib.

3 Local Search, Tabu Search, and Life Span Method

In this section, we briefly describe local search and tabu search, and introduce a variant of tabu search called the *Life Span Method* on which the algorithm that we will present is based. We describe the outline of these algorithms in terms of the generic combinatorial optimization problem.

3.1 Combinatorial Optimization Problem

A generic combinatorial optimization problem is defined as follows [25].

Let B be a finite set called the *ground set*. The objective of the combinatorial optimization problem is to find a minimum cost element in the set of feasible solutions $X \subseteq 2^B$, i.e.,

$$\min\{c(x) : x \in X\},$$

where $c : X \to \Re$ denotes a cost mapping.

Given a feasible solution x in a particular problem, we can define a set of solutions $N(x)$ that are "close" to it in a sense. We call $N(x)$ the *neighborhood* of x.

Given a combinatorial optimization problem, a mapping

$$N : X \to 2^X$$

is called the neighborhood.

We want to find a *global optimum*, which is a solution with the minimum possible cost. Finding a global optimum can be prohibitively difficult, but it is often possible to find a solution x which is best in the sense that there is nothing better in its neighborhood $N(x)$. We call the solution in which none of its neighbors has a lower cost a *local optimum*.

3.2 Local Search

We first review local search to understand tabu search and the life span method. Given a neighborhood $N : X \to 2^X$, the mapping *improve* used in local search is defined by

$$improve(x) = \begin{cases} \text{any } x' \in N(x) & \text{with } c(x') < c(x) \text{ if such an } x' \text{ exists} \\ \emptyset & \text{otherwise.} \end{cases}$$

Using this mapping, a prototype of local search algorithm is described as follows.

```
procedure local search
  1 x := some initial feasible solution
  2 while improve(x) ≠ ∅ do
  3     x := improve(x)
  4 return x
```

Figure 1: Local Search.

A good survey of local search procedures can be found in [25, § 18].

Although many variants of local search have been proposed, we adopt tabu search (or steepest ascent mildest descent method) introduced by Glover [13, 14] and independently by Hansen [17, 18] , as a basic ingredient for designing our algorithm. The reason is that tabu search is simpler and more efficient than other metastrategies such as the simulated annealing algorithm [1, 4, 29] and the genetic algorithm [16, 24].

3.3 Tabu Search

The main idea of tabu search is to use the *best* neighbor instead of an *improved* neighbor, and to forbid some moves to avoid cycling. Here, a *move* is a pair of solutions (x, x') such that $x \in X$ and $x' \in N(x)$. The set of solutions forbidden to be visited again is stored in the so-called *tabu list* TL. The tabu search algorithm uses a mapping *best* which can be defined by

$$best(x) = \begin{cases} x' & \text{if } c(x') \leq c(y) \text{ for all } y \in N(x) \setminus TL \\ \emptyset & \text{if } N(x) \setminus TL = \emptyset. \end{cases}$$

Using the above terminology, a prototype of tabu search can be described as follows:

procedure tabu search
1 $t := 0$ $/*$ t represents the number of iterations $*/$
2 $x_0 :=$ some initial solution
3 $TL := \emptyset$ $/*$ TL represents the tabu list $*/$
4 $tabulength :=$ a positive integer
5 **while** stopping-criterion $\neq$ yes **do**
6 $\quad x_{t+1} := best(x_t)$
7 $\quad TL := TL \cup \{x_t\} \setminus \{x_{t-tabulength}\}$
8 $\quad t := t + 1$
9 **return** x

Figure 2: Tabu Search.

3.4 Life Span Method

In some applications, it is very time-consuming to store the solutions in the tabu list; so Glover recommended the following approximation. An *attribute* is the 'coding' or 'fingerprint' of move (x, x') of solutions. More precisely, we assume that there exists a mapping $\psi : X \times X \to \mathcal{A}$, where $\mathcal{A}$ denotes the set of attributes. When a solution x is moved to the new one $x' \in N(x)$, we store attribute $\psi(x', x)$ in the tabu list to avoid a move from x' to x. Then, move (x, x') cannot be used if attribute $\psi(x, x')$ is in the tabu list. For more details, see [13, 14].

The Life Span Method (LSM) is a variant of tabu search introduced by the authors in order to overcome some drawbacks and vagueness of the original tabu search. The main differences between the LSM and tabu search are

1. the definition of attributes;
2. the representation of tabu list;
3. the permission of infeasible solutions;

4. the basic philosophy to avoid many *ad hoc* rules and parameters.

The LSM works on 2^B instead of X, where B is the *ground set*. Solutions which are not in the feasible solution set X are also allowed to be searched. Although some tabu search algorithms in the literature have adopted such an infeasible solution approach, the LSM treats the infeasibility of solutions in an explicit way. The definition of the attribute in the original tabu search was rather vague and problem dependent. In the LSM, the set of attributes $\mathcal{A}$ corresponds to 2^B. Recall that $X \subseteq 2^B$. Given two solutions $x, x' \in 2^B$, the symmetric difference $x \triangle x' = (x' \setminus x) \cup (x \setminus x')$ is also in 2^B. Thus, the mapping ψ is simply stated as

$$\psi(x, x') = x \triangle x'.$$

For each element β of B, we define the 'Life Span' of β as the remaining iterations that β is forbidden, and denote it by $LS(\beta)$. When a solution x is moved to a new one $x' \in N(x)$, we set $LS(\beta)$ to a positive integer *tabulength* for every $\beta \in x \triangle x'$. For every iteration, we decrease $LS(\beta)$ by one if $LS(\beta) > 0$. If $LS(\beta)$ is positive, all moves (x, x') whose symmetric differences contain β are forbidden.

As in tabu search, we move to the best neighbor. Since we allow visiting infeasible solutions in the course of the algorithm, the neighborhood mapping N is defined as

$$N : \tilde{X} \to 2^{\tilde{X}},$$

where $\tilde{X} = 2^B$ is the set of (feasible or infeasible) solutions and the mapping *best* in tabu search is modified as

$$best(x) = \arg\min\{c(y) : y \in N(x) \text{ such that } LS(\beta) = 0 \text{ for all } \beta \in x \triangle y\}.$$

Now a prototype of the LSM is described as follows.

```
procedure life span method
 1 x := some initial solution
 2 LS(β) := 0 for all β ∈ B
 3 tabulength := a positive integer
 4 while stopping-criterion ≠ yes do
 5     x' := best(x)
 6     LS(β) := tabulength for all β ∈ x△x'
 7     x := x'
 8     LS(β) := LS(β) − 1 for all β ∈ B such that LS(β) > 0
 9 return x
```

Figure 3: Life Span Method.

The LSM has the following merits.

1. We can determine the attributes without any ambiguity.

2. Checking the tabu status can be done in $O(1)$ time in the LSM, while the queue implementation recommended by Glover [13, 14] requires $O(tabulength)$ time to do the same operation. Instead, the LSM requires an additional $O(|B|)$ memory which creates no problem in almost all applications.

3. The LSM has more flexibility. For example, we can randomize *tabulength* to diversify the search.

4. Allowing infeasible solutions makes it possible to escape from local optima.

Not only are the mathematical definitions between tabu search and the LSM different, but the fundamental philosophy is also different. The philosophy of tabu search is to collect principles of intelligent problem solving [15]; so the parameters to control the algorithm may be very large. Meanwhile, our philosophy is to keep the number of control parameters as small as possible. The details of the LSM can be found in the companion paper [21].

4 Life Span Method for the Quadratic Assignment Problem

In this section, we explain the LSM for the QAP.

4.1 Neighborhood

By denoting the set of all permutation by Π, we can define the neighborhood as follows,

Definition 2 (2-opt neighborhood for QAP)
Given a permutation π, a 2-opt neighbor N is defined by

$$N_2(\pi) = \{\pi' \in \Pi : \pi'(i) = \pi(j), \pi'(j) = \pi(i), \pi'(k) = \pi(k)(k \neq i, j) \text{ for all } i, j \in V, i \neq j\}.$$

Chakrapani and Skorin-Kapov [7], Taillard [28], and many other researchers used the 2-opt neighborhood.

Definition 3 (3-opt neighborhood for QAP)
Given a permutation π, a 3-opt neighbor N is defined by

$$N_3(\pi) = \{\pi' \in \Pi : \pi'(i) = \pi(j), \pi'(j) = \pi(k), \pi'(k) = \pi(i), \pi'(\ell) = \pi(\ell) \\ (\ell \neq i, j, k) \text{ for all } i, j, k \in V, i \neq j \neq k\}.$$

4.2 Attribute Set and Life Span

In usual tabu search implementations, the attribute set should be determined by considering the structure of the problem to be solved. In the LSM, the attribute set is determined without any ambiguity; the attribute set is defined on the *ground set*. The *ground set* B of the QAP corresponds to the Cartesian product the set of objects (facilities) and the set of

locations, i.e., $B = V \times V$. Recall that we used the $0-1$ variable x_{ij} which is set to 1's if object i is assigned to location j in our 'natural' formulation of the QAP Thus, an element of the ground set of the QAP is the pair of an object and its location. An attribute mapping ψ corresponds to the symmetric difference of two permutations.

The set X of feasible solutions is the set of permutation matrices which is a subset of 2^B, i.e., $X \subseteq 2^{V \times V}$. We denote the permutation matrix ($n \times n$ square $0-1$ matrix whose row and column sums are all $1's$) associated with permutation π by M_π. Given two permutations π and π', we define a set $\mathcal{M}(\pi \setminus \pi')$ as follows: $(i,j) \in \mathcal{M}(\pi \setminus \pi')$ if and only if $M_\pi(i,j) = 1$ and $M_{\pi'}(i,j) = 0$. The symmetric difference of two permutations π and π' is $\mathcal{M}(\pi \setminus \pi') \cup \mathcal{M}(\pi' \setminus \pi)$.

When we move from π to π', we set $LS(i,k)$ to a positive integer, *tabulength*, for all $(i,k) \in \mathcal{M}(\pi \setminus \pi')$, and *tabulength'*, for all $(i,k) \in \mathcal{M}(\pi' \setminus \pi)$. The life span LS is reduced by 1 for each iteration, and object i is prohibited to move to location k again while $LS(i,k)$ is positive. In our implementation, we always set $tabulength' = 0$ to simplify the parameter tuning. In the sequel, we call our attribute the *object-location* attribute.

The previous implementation of tabu search [6, 7, 26, 27, 28] adopted the set of pairs of two objects i, j as an element of the attribute set. We henceforth call this attribute *the object pair* attribute. For the 2-opt move that swaps the positions of two objects i and j, we set $LS(i,j)$ to a positive integer, *tabulength*. For the 3-opt move that exchanges the positions of three objects i, j and k, we set $LS(i,j) = LS(j,k) = LS(k,i)$ to *tabulength*.

We will compare these two attributes (the object-facility attribute and the *object pair* attribute) in Section 5.2.4). The experimental results will show the advantage of our attribute selection.

4.3 Calculation of the Difference in Costs

Here, we consider an efficient procedure to compute the difference $c(\pi') - c(\pi)$ in costs.

We first consider the 2-opt neighborhood. The difference Δ_{ij} when we swap two objects i and j can be computed as follows [5]:

$$\Delta_{ij} = \sum_k (f_{jk} - f_{ik})(d_{\pi(i)\pi(k)} - d_{\pi(j)\pi(k)}).$$

Using this formula, a straightforward implementation of the LSM requires $O(n^3)$ time per iteration. If we choose the best element among $O(n^2)$ candidate pairs of two objects, the average (amortized) computational requirements can be reduced to $O(1)$ per pair. This can be done using the additional $O(n^2)$ memory requirements. We first rewrite the Δ_{ij} as follows:

$$\begin{aligned} \Delta_{ij} &= \sum_k (f_{jk} - f_{ik})(d_{\pi(i)\pi(k)} - d_{\pi(j)\pi(k)}) \\ &= \sum_k (f_{jk} d_{\pi(i)\pi(k)} - f_{jk} d_{\pi(j)\pi(k)} + f_{ik} d_{\pi(j)\pi(k)} - f_{ik} d_{\pi(i)\pi(k)}). \end{aligned}$$

If we store the value $\delta_{ip} = \sum_k f_{ik} d_{p\pi(k)}$ for every i and p which represents the difference in cost when object i is moved to the location p, Δ_{ij} can be computed in $O(1)$ time as follows:

$$\Delta_{ij} = \delta_{j\pi(i)} - \delta_{j\pi(j)} + \delta_{i\pi(j)} - \delta_{i\pi(i)} + 2 f_{ij} d_{\pi(i)\pi(j)}. \tag{5}$$

Initially, we calculate all δ's in $O(n^3)$ time. If two objects a and b swap their positions, the value δ_{ip} is recomputed as follows:

$$\delta_{ip} := \delta_{ip} + (f_{ib} - f_{ia})(d_{\pi(i)\pi(a)} - d_{\pi(i)\pi(b)}).$$

Since each updating can be done in $O(1)$ time, and the number of updates is $O(n^2)$, we can update the array δ in $O(n^2)$ time. Finding the minimum of Δ_{ij} can be done in $O(n^2)$ time; so one iteration of the LSM is $O(n^2)$, which is an $O(n)$ refinement of the naive implementation. This technique was used in Skorin-Kapov's tabu search [26].

We extend this technique to the 3-opt neighborhood. We can compute the difference Δ_{ijh} in costs when we swap three objects i, j, h using the following formula:

$$\begin{aligned}\Delta_{ijh} &= \delta_{i\pi(j)} - \delta_{i\pi(i)} + \delta_{j\pi(h)} - \delta_{j\pi(j)} + \delta_{h\pi(i)} - \delta_{h\pi(h)} \\ &\quad + f_{ih}(d_{\pi(j)\pi(i)} - d_{\pi(j)\pi(h)}) \\ &\quad + f_{ji}(d_{\pi(h)\pi(j)} - d_{\pi(h)\pi(i)}) \\ &\quad + f_{hj}(d_{\pi(i)\pi(h)} - d_{\pi(i)\pi(j)}) \\ &\quad + f_{ij}d_{\pi(i)\pi(j)} + f_{jh}d_{\pi(j)\pi(h)} + f_{ih}d_{\pi(i)\pi(h)}. \end{aligned} \tag{6}$$

If three objects a, b, and c swap their positions, the value δ_{ip} is recomputed as follows:

$$\delta_{ip} := \delta_{ip} + (f_{ib} - f_{ic})(d_{\pi(i)\pi(c)} - d_{\pi(i)\pi(b)}) + (f_{ic} - f_{ia})(d_{\pi(i)\pi(a)} - d_{\pi(i)\pi(b)}).$$

Thus, we can find the best move in the 3-opt neighborhood in $O(n^3)$ time.

4.4 Long Term Memory

We also use the long term memory [13] to avoid cycling. The long term memory is used to diversify the search compelling regions that are not visited before. Note that the similar idea was used in the classical local search literature [25].

We use a pair of object i and location k for the long term memory LTM as in the life span LS. When object i is moved from location k, we increase $LTM(i, k)$ by 1.

A diversification factor $\alpha \times LTM(i, k)/100$ is added to the objective function $c(\pi)$, i.e., when we change permutation π to permutation π', we use the perturbated objective function $\hat{c}(\pi')$ defined as

$$\hat{c}(\pi') = c(\pi') + \sum_{(i,k)\in\mathcal{M}(\pi\setminus\pi')} LTM(i, k). \tag{7}$$

For the 2-opt neighborhood, the difference $\Delta'(i, j)$ of two function values $\hat{c}(\pi')$ and $\hat{c}(\pi)$ becomes as follows:

$$\Delta'(i, j) := \Delta(i, j) + \alpha \times (LTM(i, \pi(j)) + LTM(j, \pi(i))/100$$

In the 3-opt neighborhood, we do not use the long term memory to simplify the implementation and the parameter settings.

4.5 Life Span Clear Strategy

To intensify the search, we periodically clear the tabu status (life span). We call this strategy the *life span clear* and abbreviate it LSC in the sequel. The LSC strategy is executed in the following way. If the number of iterations exceeds the predetermined value $Clear\_Count$ without improving the best value obtained so far, we continue the search from the current solution after setting life span $LS(i,k)$ to 0 for all i,k in V.

4.6 Change and Termination Criterion

We use two neighborhoods, 2-opt and 3-opt, as follows. We first use the 2-opt neighborhood. When the predetermined number of LSC's $Change23\_Count$ is exhausted without improving the objective function value, we restart the search from the 'best' solution using both the 2-opt and 3-opt neighborhoods.

Similarly, we restart the search from the 'current' solution with the 2-opt neighborhood when the number of LSC's exceeds the predetermined value $Change32\_Count$ without improving the best value obtained.

We define the termination criteria as follows. When the predetermined number of iterations, $Stop\_Count$, throughout 2-opt and 3-opt phases is exhausted without improving the objective function value, the algorithm stops.

Figure 4 shows the LSM for the QAP that incorporates the 2-opt and 3-opt neighborhoods, life span (attribute), long term memory, and life span clear strategies.

5 Numerical Experiments

In this section, we report the results of our computational experiments. All computational experiments were executed on **Hitachi 3050** and algorithms were coded in the C language. Running time were measured by making the system call **times** and converting to seconds.

Since parameter tunings would be of crucial importance, we executed extensive experiments to select good or appropriate parameters for the LSM in Section 5.2. The major difference between the previous tabu search algorithms and our LSM is the definition of the attributes; so we compare these two alternative definitions of the attributes in Section 5.3. Then, we compare the performance of the LSM with that of tabu search by Chakrapani and Skorin-Kapov [7] that was known to be one of the best heuristic algorithms.

5.1 Test Problems

In our experiments, we used the same problems which have been used by Chakrapani and Skorin-Kapov [7] to compare the performance with her tabu search (*Augmented Par_tabu*). These problems are included in QAPLIB [2] (see Section 2.3).

Table 1 summarizes the experimental results in the previous work. All best solutions were found by tabu search [7] or the genetic hybrid [10].

Table 1: Best known upper and lower bounds of Skorin-Kapov's benchmark instances.

Problem	n	Best upper bound	Best lower bound
sko42	42	15812 [7]	14934 [20]
sko49	49	23386 [7]	22004 [20]
sko56	56	34458 [7]	32610 [20]
sko64	64	48498 [7]	45736 [20]
sko72	72	66256 [7]	62691 [20]
sko81	81	90998 [10]	86072 [20]
sko90	90	115534 [7]	108493 [20]
sko100a	100	152002 [10]	142668 [20]
sko100b	100	153890 [10]	143872 [20]
sko100c	100	147862 [10]	139402 [20]
sko100d	100	149576 [10]	139898 [20]
sko100e	100	149150 [10]	140105 [20]
sko100f	100	149036 [10]	139452 [20]

Table 2: The recommended values of the LSM parameters.

Prob. name	$tabulength$	α	$Clear\_Count$	$Change23\_Count$	$Change32\_Count$	$Stop\_Count$
$sko42$	15	25	100	10	5	5000
$sko49$	20	25	100	10	5	5000
$sko56$	25	30	100	10	5	5000
$sko64$	30	35	100	10	5	5000
$sko72$	35	40	100	20	5	10000
$sko81$	40	40	100	40	5	20000
$sko90$	45	45	100	80	10	40000
$sko100a$	50	50	100	100	10	50000
$sko100b$	50	50	100	100	10	50000
$sko100c$	50	50	100	100	10	50000
$sko100d$	50	50	100	100	10	50000
$sko100e$	50	50	100	100	10	50000
$sko100f$	50	50	100	100	10	50000

procedure life span method for QAP.
1 select $\pi(\in \Pi)$ arbitrary
2 $LS(i,j) := 0$, $LTM(i,j) := 0$ for all $(i,j) \in V \times V$
3 **while** terminate-criterion $\neq$ yes **do**
4 **while** change23-criterion $\neq$ yes **do**
5 $\pi^* := \arg\min\{\hat{c}(\pi') : \pi' \in N_2(\pi), LS(i,k) = 0 \text{ for all } (i,k) \in \mathcal{M}(\pi' \setminus \pi)\}$
6 $LS(i,k) := tabulength$ for all $(i,k) \in \mathcal{M}(\pi \setminus \pi^*)$
7 $LTM(i,k) := LTM(i,k) + 1$ for all $(i,k) \in \mathcal{M}(\pi \setminus \pi^*)$
8 $\pi := \pi^*$
9 **for all** $(i,j) \in V \times V$
10 **if** $LS(i,j) > 0$ **then** $LS(i,j) := LS(i,j) - 1$
11 **while** change32-criterion $\neq$ yes **do**
12 $\pi^* := \arg\min\{c(\pi') : \pi' \in N_3(\pi) \cup N_2(\pi), LS(i,k) = 0 \text{ for all } (i,k) \in \mathcal{M}(\pi' \setminus \pi)\}$
13 $LS(i,k) := tabulength$ for all $(i,k) \in \mathcal{M}(\pi \setminus \pi^*)$
14 $\pi := \pi^*$
15 **for all** $(i,j) \in V \times V$
16 **if** $LS(i,j) > 0$ **then** $LS(i,j) := LS(i,j) - 1$
17 **return** π

Figure 4: The LSM for the QAP.

5.2 Parameter Optimization

In this section, we will describe the experiments that induce the parameter settings of our experiments. The recommended parameters that we will derive via extensive experiments would be of value when we try to find an appropriate parameter setting for new data sets.n Table 2 shows the recommended parameters. Also, the process for finding good parameters would be of importance because it gives us the insight of the roles of various parameters.

The experiments to compare the performance of two alternative attributes (the *object-location* and *object pair* attributes) will be shown in Section 5.2.4.

5.2.1 Optimization of *tabulength*

Since *tabulength* is one of the most important parameters which affect on the performance and how mush time is required in searching, we set *tabulength* with care at first.

We performed 10 independent runs from different (random) initial solutions for each *tabulength*. In this set of experiments, we do not use the other strategies such as the long term memory, 3-opt neighborhood, and life span clear (LSC). We set the other parameters as follows:

$$(\alpha, Clear\_Count, Change23\_Count, Change32\_count, Stop\_Count) = (0, 0, 0, 0, 5000).$$

Figure 5 shows the relationship between *tabulength* and the best cost function value obtained. Table 3 shows, for each of a set of values for *tabulength*, the average cost and its

Table 3: Average cost and its variance by changing *tabulength* (* indicates the proper range).

range	Average Cost	Variance	range	Average Cost	Variance
$1-10$	16129.9	12496	$151-160$	16201.8	10700.7
$11-20*$	16110.2	5367.8	$161-170$	16227.8	11819.6
$21-30$	16134.6	4668.7	$171-180$	16232.8	11601.5
$31-40$	16147.5	6214.4	$181-190$	16189.8	10854.5
$41-50$	16155.5	6603.1	$191-200$	16183.1	9377.9
$51-60$	16181.6	5579.9	$201-210$	16210.8	8341.6
$61-70$	16196.3	5762.9	$211-220$	16229.8	12124.0
$71-80$	16164.1	6058.8	$221-230$	16208.0	7358.0
$81-90$	16184.0	8463.3	$231-240$	16219.7	13133.3
$91-100$	16216.4	11752.5	$241-250$	16241.1	12142.9
$101-110$	16223.1	8565.0	$251-260$	16202.6	12281.3
$111-120$	16179.8	8510.8	$261-270$	16221.9	9151.1
$121-130$	16205.8	9561.6	$271-280$	16192.4	13206.8
$131-140$	16231.4	8927.1	$281-290$	16227.4	9497.3
$141-150$	16204.8	9745.1	$291-300$	16223.2	9715.4

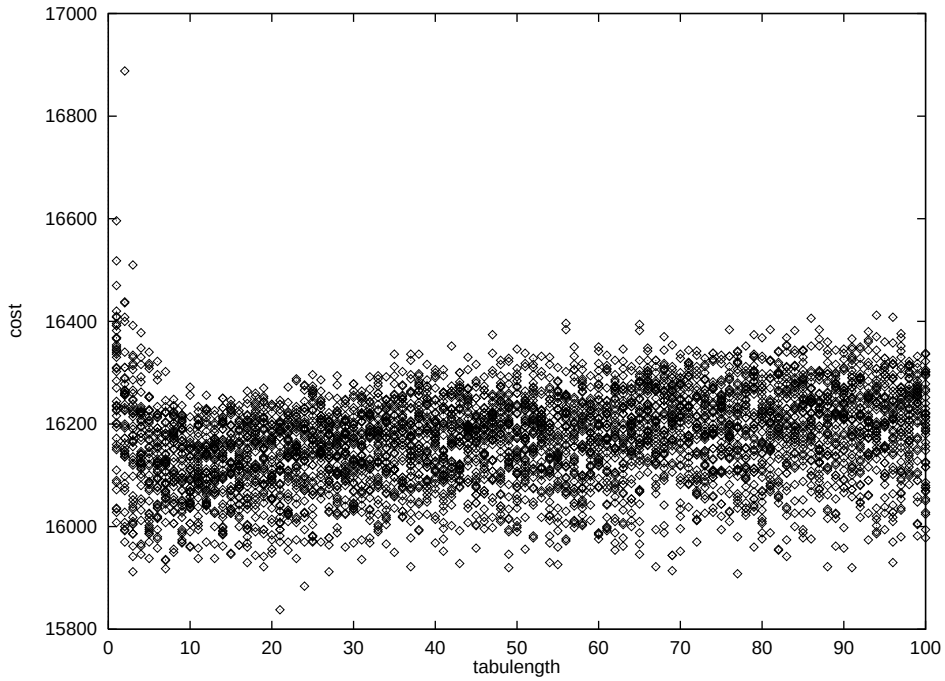

Figure 5: The relationship between the cost function and *tabulength* : ($Stop\_Count = 5000$).

Table 4: Average cost and its variance by changing α (* indicates the proper range).

Range	Average Cost	Variance	Range	Average Cost	Variance
0	16185.4	8764.3	$51-60$	16145.1	5508.4
$1-10$	16165.3	5192.9	$61-70$	16158.9	4233.8
$11-20$	16137.7	6314.6	$71-80$	16141.6	7088.2
$21-30^*$	16136.7	7782.2	$81-90$	16154.9	8621.3
$31-40$	16144.0	6526.0	$91-100$	16136.9	6580.0
$41-50$	16159.0	5447.9	-	-	-

variance.

Table 3 indicates that the average cost first decreases, then increases as *tabulength* increases and all ranges have high variances. The same behavior occurs for the other test problems. Observe that there is a *proper* range which gives the best average cost, and our recommended value of $tabulength = 15$ falls within this range.

By executing similar experiments on other test problems, appropriate values of *tabulength* were obtained as in Table 3. For the 3-opt neighborhood, we used the same values of *tabulength* obtained by the experiments of the 2-opt neighborhood because we cannot find major differences in performance when we change the parameter for the 3-opt neighborhood and we have no reason to replace the parameter settings for the 2-opt neighborhood.

5.2.2 Optimization of α

The remaining important parameter is α that controls the intensity of the long term memory. We set the other parameters as follows:

$$(tabulength, Clear\_Count, Change23\_Count, Change32\_count, Stop\_Count) = (15, 0, 0, 0, 5000).$$

We performed 10 runs from different initial random solutions for each α. Figure 6 shows the relationship between α and the best cost function value obtained. Table 4 shows the results of this experiments. When α is between 21 and 30, the average cost is smaller than the other ranges, but as α becomes too large or 0, i.e., we do not use the long term memory, the average cost becomes larger.

Since the long term memory increases until the algorithm terminates, the gap between $\Delta(i, j)$ and $\Delta'(i, j)$ becomes larger as the number of iterations become larger. As a result, the long term memory may demolish the 'true' objective function.

5.2.3 Optimization of $Clear\_Count$

The last important parameter to be analyzed is $Clear\_Count$. We set the other parameters as follows:

$$(tabulength, \alpha, Change23\_Count, Change32\_count, Stop\_Count) = (15, 25, 0, 0, 5000).$$

Figure 8 shows the relationship between $Clear\_Count$ and the best cost function value obtained. We performed 10 runs from different initial (random) solutions for each α. Table

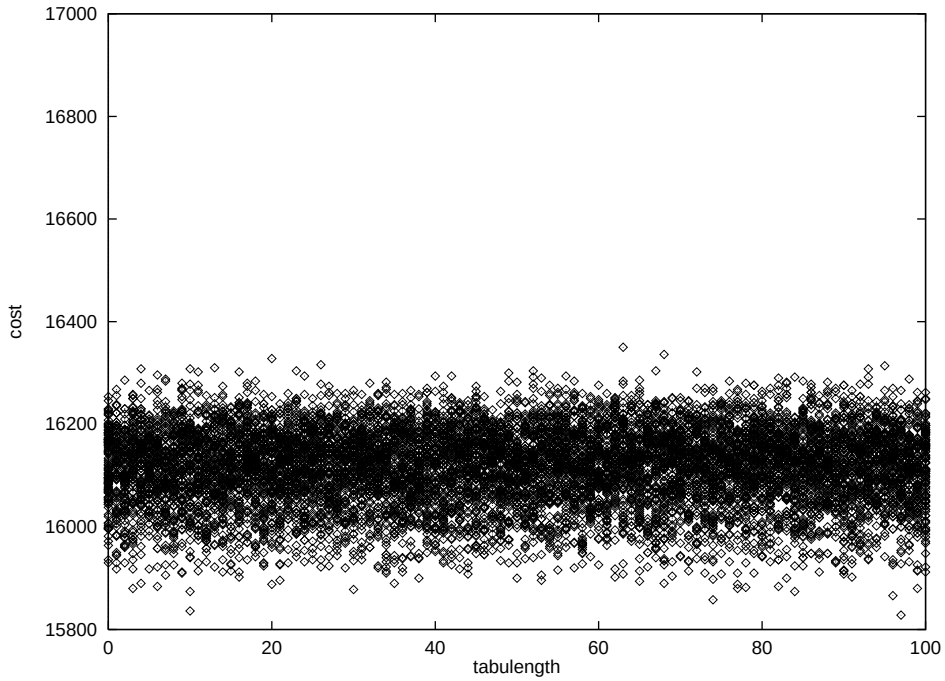

Figure 6: The relationship between the cost function and α : ($tabulength = 15, Stop\_Count = 5000$).

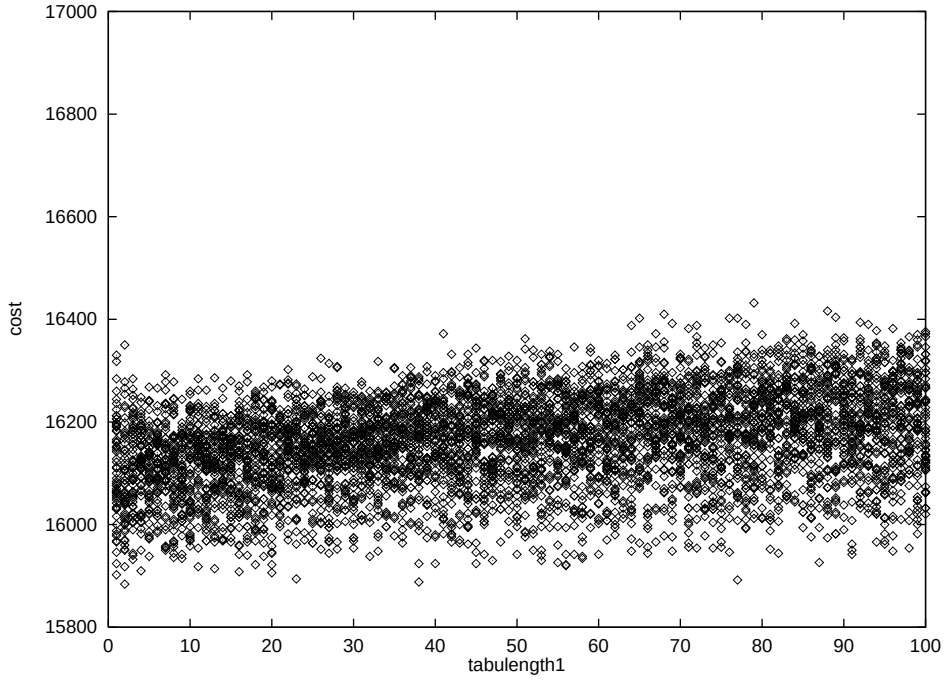

Figure 7: The relationship between the cost function and $tabulength$: ($\alpha = 10, Stop\_Count = 5000$).

Table 5: Average cost and its variance by changing $Clear\_Count$ (* indicates the proper range).

Range	Average Cost	Variance	Range	Average Cost	Variance
100*	15869.5	278.8	$1001-1100$	15974.5	7898.8
200	15903.0	635.0	$1101-1200$	15964.5	4430.8
300	15904.0	262.0	$1201-1300$	15963.0	2115.0
400	15898.5	776.8	$1301-1400$	15995.0	2777.0
500	15965.0	1075.0	$1401-1500$	15933.0	2219.0
600	15924.5	426.8	$1501-1600$	16010.5	1802.8
700	15933.5	2164.8	$1601-1700$	15972.0	3944.0
800	15968.0	1462.0	$1701-1800$	16017.5	2852.8
900	15944.5	980.8	$1801-1900$	16018.0	996.0
1000	15990.0	3924.0	$1901-2000$	15965.0	821.0

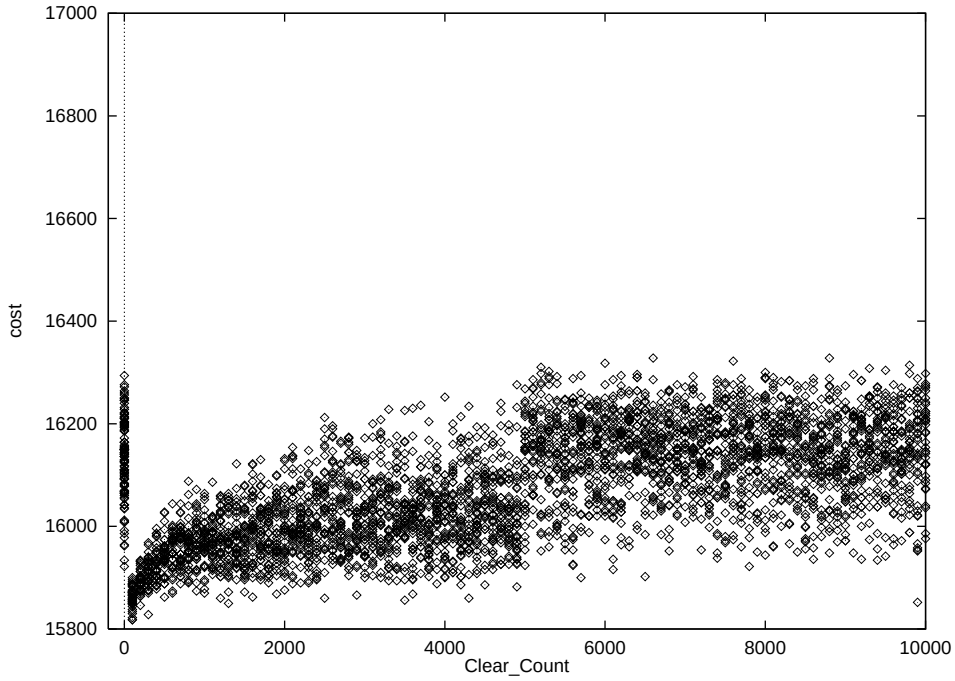

Figure 8: The relationship between the cost function and $Clear\_Count$: ($tabulength = 15, \alpha = 10, Stop\_Count = 5000$).

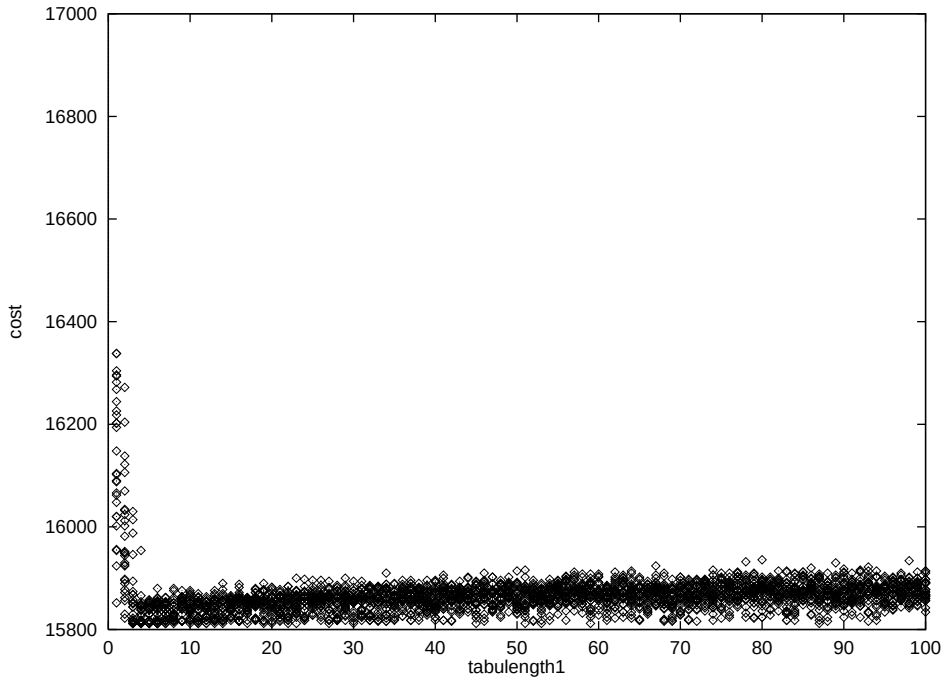

Figure 9: The relationship between the cost function and *tabulength* : ($\alpha = 0, Clear\_Count = 100, Stop\_Count = 5000$).

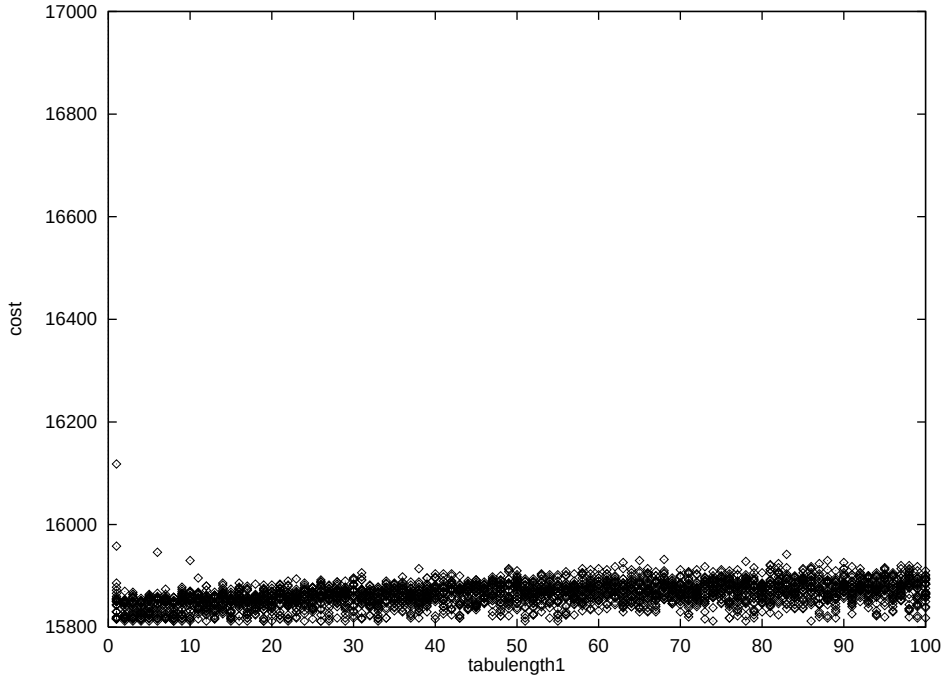

Figure 10: The relationship between the cost function and *tabulength* : ($\alpha = 10, Clear_Count = 100, Stop\_Count = 5000$).

Table 6: Average cost and its variance incorporating all strategies (* indicates the proper range).

Range	Average Cost	Variance	Range	Average Cost	Variance
$1-10$	15833.4	377.6	$151-160$	15880.6	218.4
$11-20^*$	15831.4	276.8	$161-170$	15884.4	235.0
$21-30$	15861.6	417.8	$171-180$	15876.0	120.8
$31-40$	15857.5	305.7	$181-190$	15869.6	767.8
$41-50$	15861.8	378.8	$191-200$	15890.0	312.8
$51-60$	15869.0	518.6	$201-210$	15870.6	315.2
$61-70$	15867.6	236.64	$211-220$	15868.0	450.4
$71-80$	15872.6	340.0	$221-230$	15876.0	326.4
$81-90$	15880.4	177.4	$231-240$	15872.8	86.56
$91-100$	15874.0	248.8	$241-250$	15882.2	221.96
$101-110$	15870.6	488.0	$251-260$	15882.8	614.56
$111-120$	15870.8	538.6	$261-270$	15884.6	192.8
$121-130$	15874.0	316.0	$271-280$	15862.6	448.0
$131-140$	15873.0	147.4	$281-290$	15883.2	401.8
$141-150$	15866.0	623.2	$291-300$	15882.8	471.4

5 shows the results of experiments, The recommended value of parameter $Clear\_Count$ is 100. For the other test problems, we find that $Clear\_Count = 100$ is a good choice via the similar set of experiments. For the 3-opt neighborhood, the appropriate range of parameter $Clear\_Count$ is $[5, 10]$ via the similar sets of experiments.

5.2.4 Experiments Incorporating All Strategies

Generally speaking, as the parameters $(Stop\_Count, Change23\_Count, Change32\_Count)$ become larger, the solution values obtained become better. We can set the values of these parameters by considering both the running time and the solution values. We performed the similar experiments as in Section 5.2.1 by incorporating all parameters optimized above. We set the parameter as follows:

$$(\alpha, Clear\_Count, Change23\_Count, Change32\_Count, Stop\_Count) = (25, 100, 100, 5, 5000)$$

We run the LSM with the above parameters starting from 10 different initial solutions for each *tabulength*. Figure 11 illustrates the relationship between *tabulength* and the best cost incorporating all strategies. By comparing Figure 5 with Figure 11, we observe that the values and variances of solutions are dramatically decreased by incorporating various strategies and then by optimizing the parameters.

5.3 Comparison of the Attribute Selection

Table 7: Performance comparison of best upper bounds (BUB) and the total number of iterations (TNI).

	LSM			*Augmented Par_tabu*	
Prob. name	BUB	TNI(2-opt)	TNI(2-opt + $n \times$ 3-opt)	BUB	TNI
sko42	15812	9653	17045	15812	89432
sko49	23386	9659	17352	23386	112810
sko56	34458	20067	26787	34458	136901
sko64	48498	28324	38628	48498	145056
sko72	66256	36097	51829	66256	198129
sko81	90998	36467	59066	91008	191571
sko90	115534	45980	63620	115586	268416
sko100a	152002	75345	126545	152014	199882
sko100b	153890	68208	123208	153890	274480
sko100c	147862	79829	141229	147868	306954
sko100d	149576	67788	118788	149596	257855
sko100e	149150	69144	124444	149156	311458
sko100f	149036	72345	120645	149036	308587

We then compare two alternative definitions of the attribute set. One is the *object-location* attribute that keeps the pair of an object and its location in the life span. The other is the *object pair* attribute that keeps the pair of objects in the life span.

We have already shown how to tune up the parameters for the *object-location* attribute that is our own choice. We execute the same experiments for the *object pair* attribute and find the recommended values of the various parameters.

$$(\alpha, Clear\_Count, Change23\_Count, Change32\_Count, Stop\_Count) = (30, 100, 100, 5, 5000)$$

Figure 12 shows the relationship between *tabulength* and the best cost for the *object pair* attribute. As can be seen from Figures 11 and 12, the *object-location* attribute is less sensitive to choices of *tabulength*, and thus yields wider proper range than the *object pair* attribute. The solution values and variances obtained by the *object-location* attribute are smaller than those of the *object pair* attribute. The same phenomenon were observed in the other test data.

5.4 Results of Experiments

In this section, we present the results of experiments using the parameters adjusted via the above experiments.

First, we compare the best found solutions with previous results. Since each researcher has different computational environment, if we can know, we substitute the total number of iterations for the executed time. Table 7 summarizes the best upper bounds and the total number of iterations achieved by the LSM and *Augmented Par_tabu* [7]. We consider that one iteration of 3-opt neighborhood corresponds to n times as large as one iteration of 2-opt neighborhood.

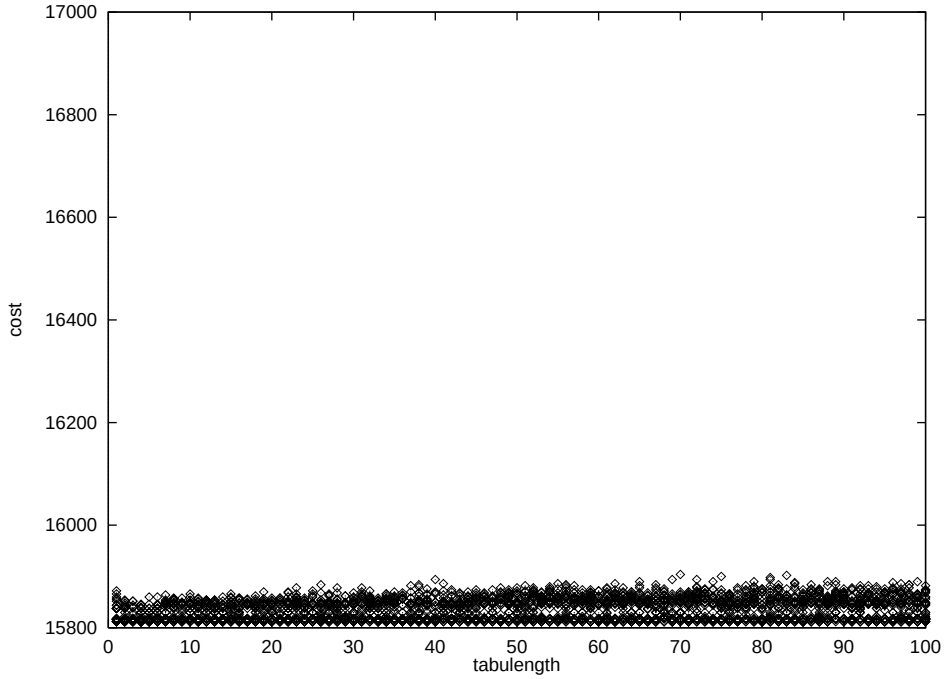

Figure 11: The relationship between the cost function and *tabulength* incorporating all strategies. The attribute is the *object-location* type.

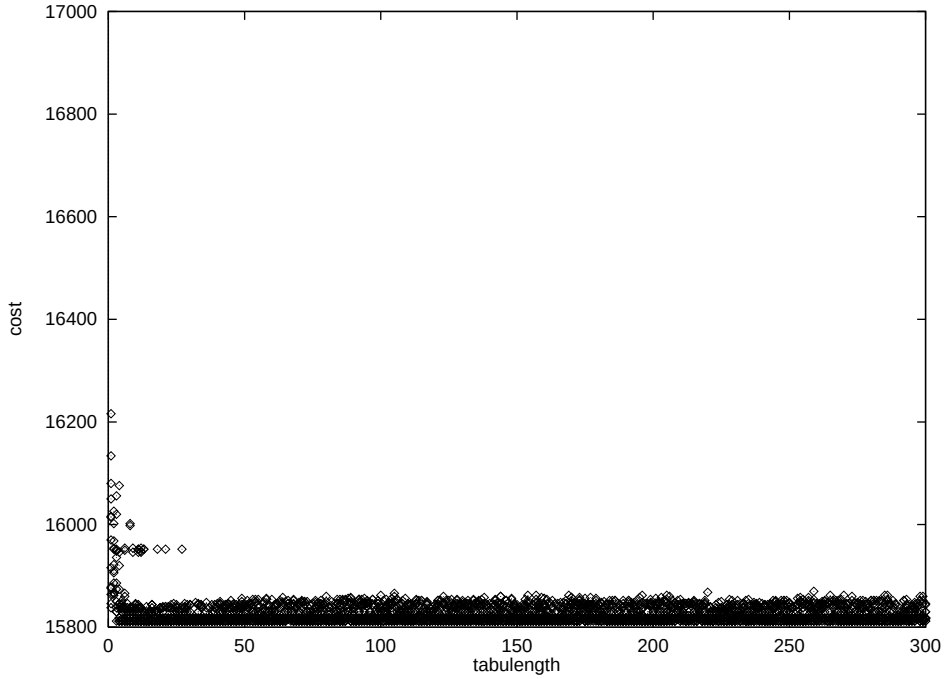

Figure 12: The relationship between the cost function and *tabulength* incorporating all strategies. The attribute is the *object pair* type.

Table 8: The average behavior of the LSM.

	Time(sec)			Solution		
Name	Min	Avg (Std. Dev.)	Max	Min	Avg (Std. Dev.)	Max
sko42	297.5	473.9(115.6)	761.5	15812	15825.3(17.0)	15864
sko49	116.9	180.1(55.0)	295.4	23386	23426.5(25.1)	23462
sko56	204.4	381.3(157.0)	728.4	34458	34518.2(39.6)	34570
sko64	372.8	512.1(121.1)	725.1	48498	48552(65.4)	48962
sko72	841.5	1128.2(250.0)	1581.9	66256	66405.2(87.4)	66550
sko81	1252.6	1765.3(464.6)	2602.7	90998	91217.2(189.6)	91450
sko90	1806.9	2752.6(638.7)	3935.3	115534	115759.8(114.38)	116268
sko100a	5360.3	5461.3(102.8)	5562.4	152002	152093.9(41.1)	152222
sko100b	5505.2	5569.9(65.9)	5634.7	153890	153943.9(41.5)	154108
sko100c	6266.0	6983.9(729.6)	7701.4	147862	147893.2(23.7)	147966
sko100d	5257.7	5294.6(37.56)	5331.6	149576	149670.8(110.4)	149972
sko100e	5565.3	5941.7(382.9)	6318.2	149150	149215.9.9(100.9)	149694
sko100f	5342.6	5493.1(153.1)	5643.6	149036	149093.3(45.9)	149216

In Table 8, we investigate the average performance of the LSM by performing 30 runs on each problem to obtain the sample mean, standard deviation, maximum and minimum of the solution values, and running time.

6 Conclusions

We applied a variant of tabu search, which we call the life span method (LSM), to the quadratic assignment problem (QAP) and performed the extensive experiments including parameter optimization. We also presented a formula to calculate the differences in objective function values for all 3-opt neighborhoods in $O(n^3)$ time.

The conclusions obtained by extensive experiments are as follows.

1. The LSM is an extremely efficient approximate algorithm for the QAP that has been known to be one of the notoriously difficult $\mathcal{NP}$-hard problems. For all instances we tested, relatively long LSM runs give us the best known solutions.

2. There can be an advantage to the combination of the 2-opt and 3-opt neighborhoods rather than the single use of the 2-opt neighborhood. The 3-opt neighborhood seems to be of value to escape from a deep local optima.

3. The *object-location* attribute that is the natural consequence of the definition of the LSM seems to be superior to the *object pair* attribute that has been used in the literature. The *object-location* attribute combined with other strategies makes the search robust; there exists a broad proper range of appropriate values of algorithm parameters.

4. The extensive experimental analysis shows that a good value of the parameter *tabulength* that controls the short-term memory of the LSM is within the range $[n/3, n/2]$ for the

QAP with size n.

5. Definitions of attributes affect the good values of parameter *tabulength*. The recommended values of *tabulength* in this study are slightly smaller than those of previous work. Chakrapani and Skorin-Kapov [7] used two ranges $[n/2, n]$ and $[n, 2n]$. Chakrapani and Skorin-Kapov [7] recommended the range $[n/2, n]$. Skorin-Kapov [27] used several values $n/6, n/4, n/3, n/2$ of *tabulength* that was dynamically changed.

6. The long term memory would be helpful to diversify the search. It makes the proper range of good values of parameter *tabulength* much wider. Good values of the intensity of the long term memory (α) is near $n/2$, i.e., the long term memory LTM is added to the objective function after multiplied by $n/200$. Proper ranges of parameter α are much wider than those of parameter *tabulength*.

7. Clearing the tabu status (life span) periodically would be of value to intensify the search. Good values of the parameter ($Clear\_Count$) to control the interval of clearing the life span seem to be 100 or less, and are independent of the size of the problem.

8. Appropriate parameters that have been tuned up via extensive experimental analyses lead to the algorithm that performs extremely well. For new data sets, good parameters can be found using the observations obtained from our experiments.

We are prepared to provide our code for solving the QAP used in the study. Interested readers can contact one of the authors.

References

[1] E.H.L. Aarts and J.H.M Korst. *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons, Chichester, U.K., 1989.

[2] R. E. Burkard, S. Karisch, and F. Rendl. QAPLIB - a quadratic assignment prblem liblary. *European Journal of Operational Research*, 55:115–119, 1991. Updated version - Feb. 1994.

[3] R. E. Burkard and F. Rendl. A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, 17:169–174, 1984.

[4] N.E. Collins, R.W. Eglese, and B.L. Golden. Simulated annealing: an annotated bibliography. *American Journal of Mathematical and Management Sciences*, 8:205–307, 1988.

[5] D. T. Connolly. An improved annealing scheme for the QAP. *European Journal of Operational Research*, 46:93–100, 1990.

[6] J. C. Chakrapani and J. Skorin-Kapov. A connectionist approach to the quadratic assignment problem. *Computers and Operations Research*, 19(3/4):287–295, 1992.

[7] J. Chakrapani and J. Skorin-Kapov. Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research*, 41:327–341, 1993.

[8] J. Chakrapani and J. Skorin-Kapov. A constructive method for improving lower bounds for a class of quadratic assignment problems. *Operations Research*, 42:837–845, 1994.

[9] G. Finke, R. E. Burkard, and F. Rendl. Quadratic assignment problems. *Annals of Discrete Mathematics*, 31:61–82, 1987.

[10] C. Fleurent and J. A. Freland. Genetic hybrids for the quadratic assignment problem. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, American Mathematical Society, 1994.

[11] A. M. Frieze and J. Yadegar. On the quadratic assignment problem. *Discrete Applied Mathmatics*, 5:89–98, 1983.

[12] P. C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *J.SIAM*, 10:305–313, 1962.

[13] F. Glover. Tabu search I. *ORSA Journal on Computing*, 1:190–206, 1989.

[14] F. Glover. Tabu search II. *ORSA Journal on Computing*, 2:4–32, 1989.

[15] F. Glover. Tabu search. A chapter in Modern Heuristic Techniques for Combinatorial Problems, 1992.

[16] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[17] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *The Congress on Numerical Methods in combinatorial Optimization*, Capri, March 1986.

[18] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiablity problem. *Computing*, 44:279–303, 1990.

[19] T. C. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.

[20] S. E. Karisch and F. Rendl. Lower bounds for the quadratic assignment problem via triangle decompositions. Technical report 28, CDLDO-40, Technishe Universität Graz, Streyrergasse 30, A-8010 Graz, Austria, February 1994.

[21] M. Kubo. *The Life Span Method – A New Variant of Local Search –*. Technical Report 1, Tokyo University of Mercantile Marine, April 1993. presented in The Institute of Statics and Mathematics on March 29, 1993.

[22] E. L. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.

[23] Y.LI, P. M. Pardalos and M. G. C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem, 1993. To appear in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*.

[24] Z. Michalewicz. *Genetic Algorithm + Data Structure = Evolution Programs*. Springer Verlag, 1992.

[25] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[26] J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990.

[27] J. Skorin-Kapov. Extensions of a tabu search adaptation to the quadratic assignment problem. Computers and Operations Research, to appear.

[28] E. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455, 1990.

[29] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Practice*. Kluwer Academic Publishers, Dordrrecht, The Netherlands, 1987.

[30] F. Rendle and H.Wolkowicz. A recipe for best semidefinite relaxation for (0,1)-quadratic programming Research Report CORR 94-7, University of Waterloo, Waterloo, Ontario, 1994.

[31] M. R. Wilherm and T. L. Ward. Solving quadratic assignment problem by 'simulated annealing'. *IIE Transactions*, 19:107–119, 1987.