The Life Span Method – A New Variant of Local Search –

Mikio KUBO[†] and Katsuki FUJISAWA[††]

[†]*Department of Information Engineering and Logistics,*
Tokyo University of Mercantile Marine,
Etsujima, Koutou-ku, Tokyo 135-8533, Japan
kubo@ipc.tosho-u.ac.jp

[††]*Department of Architecture and Architectural Systems,*
Kyoto University,
Yoshida-Honmati, Sakyouku, Kyoto 606-8501, Japan
fujisawa@is-mj.archi.kyoto-u.ac.jp

Received April 25, 1996
Revised August 1, 1997

In this paper, we present a variant of local search, namely the Life Span Method (LSM), for generic combinatorial optimization problems. The LSM can be seen as a variation of tabu search introduced by Glover [18, 19]. We outline applications of the LSM to several combinatorial optimization problems such as the maximum stable set problem, the traveling salesman problem, the quadratic assignment problem, the graph partitioning problem, the graph coloring problem, and the job-shop scheduling problem.

Key words: local search, tabu search, combinatorial optimization, heuristic algorithms.

1. Introduction

Among many heuristic algorithms, *local search* has been known as a useful tool for solving combinatorial optimization problems. Local search is based on a simple strategy; if there exists an improved solution near the current feasible solution, we move to the improved one, and continue the process until no improved solution can be found. This strategy has an analogy to the man climbing a mountain in the darkness (this is the reason local search is sometimes termed as the hill-climbing method); he looks around him using a lantern and if he can find a higher point, he walks to that place, and repeats this process until he reaches to the point where no higher place cannot be found around him. If he is lucky, he can stand the top of the mountain; but this is not always true. He sometimes stops at a small hill, not the top of the mountain! In its very nature, all local search algorithms own some desirable characteristics such as flexibility, speed, and ease of implementation. But the defect of the mountain climber's strategy also holds for local search. Our goal is not to find a hill but to reach the top. Recently several variants of local search have been developed to overcome this defect. These variants include the simulated annealing algorithm [1, 7, 10, 27, 28, 30, 45], the genetic algorithm [23, 25, 35, 36], neural-net approach [26], and tabu search [3, 18, 19, 20, 21, 22]. In this paper, we derive a new variant of local search called the Life Span Method (LSM), which is much simpler than many other variants of local

search.

2. Local Search and Tabu Search

In this section, we briefly review local search and tabu search on which our new algorithm is based. Here, we deal with a general combinatorial optimization problem.

Definition 1 (Combinatorial Optimization Problem)
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.

The local search is based on the adjacency relation between two feasible solutions. The concept of adjacency is defined more precisely as follows.

Definition 2 (Neighborhood)
Given a combinatorial optimization problem, a mapping

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

is called the neighborhood.

Given a neighborhood $N : X \to 2^X$, a mapping *improve* used in local search is defined by

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

Then a prototype of local search algorithm is described as follows.

procedure local search
1 $x :=$ some initial feasible solution
2 **while** $improve \neq \emptyset$ **do**
3 $\quad x := improve(x)$
4 **return** x

A good survey of the local improvement procedure can be found in [39] which includes several applications.

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, the *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 used is stored in the tabu list TL and $N(x) \setminus TL$ is the set of solutions except solutions are forbidden to be used by the 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 simple tabu search can be described as follows.

procedure simple tabu search
1 $t := 0$ /* t represents the number of iterations */
2 x_0 := some initial feasible solution
3 $TL := \emptyset$ /* TL represents the tabu list */
4 $tabulength$:= a positive integer
5 **while** stopping-criterion $\neq$ yes and $best(x_t) \neq \emptyset$ **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

Note that this prototype of tabu search is extremely simple and limited. But we do not want to provide fuller characterizations of the method because we want to focus on a simple subset of tabu search. Our goal of this paper is to offer a way to re-express the basic content of this subset of ideas to make them convenient to apply. In the original implementation of tabu search, a queue with a finite length *tabulength* is merely one of the types of memory designs proposed. The element in the tabu list stored in the *tabulength* iteration is overwritten by a new attribute, i.e., the solution in the t-th iteration is stored in $TL(t \bmod tabulength)$. In some applications, it is very time-consuming to store the solutions themselves in the tabu list; so Glover recommended the following approximation.

The *attribute* is a 'coding' or 'finger-print' of a pair (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. Then, if attribute $\psi(x, x')$ is in the tabu list, move (x, x') cannot be used, i.e., the move is 'tabu', for *tabulength* iterations. Some heuristics to diversify and to intensify the search were proposed. For more details, see [18, 19, 21, 22].

3. Life Span Method

In this section, we describe a new variant of local search named the Life Span Method (LSM).

3.1. Origin

Before describing our algorithm, we mention the origin of the LSM. Our first aim was to compare some variants of local search in a fair manner. When we started the comparison, many 'modern' variants of local search had been presented by many researchers; but little attention had been paid to 'fair' comparisons of the algorithms. We had already known that some classical local search algorithms work very well; but we knew little about the newly developed algorithms such as the simulated annealing, genetic, neural net, and tabu search algorithms. For the former algorithm, Johnson et al. [27, 28] did extensive numerical experiments and concluded that the simulated annealing algorithm is not a panacea, but a neat tool which works well on some combinatorial optimization problems if we are allowed to use a large amount of computational requirements.

We first decided to do extensive experiments on tabu search, and did a series of numerical experiments on some combinatorial optimization problems. At first, we selected two graph

theoretical combinatorial optimization problems, the graph partitioning and graph coloring problems, as a test bed because we could use the same instances used in the extensive experiments due to Johnson et al. [27, 28] in which they compared some classical algorithms with the simulated annealing algorithms.

Our original implementation of tabu search was competitive with the simulated annealing algorithm, but did not significantly outperform the simulated annealing nor other competitives. To make tabu search work much more efficient, we needed some modifications of the original tabu search.

Firstly we observed that the definition of 'attributes' is a crucial factor for a good implementation of tabu search, but we could find little guideline to determine the 'good' attribute in the literature; the definitions of attributes are problem dependent, and *ad hoc* techniques were used to select probably good attributes. So we decided to use more simple and definite rule to define the attribute, which will be explained more precisely later.

The original tabu search began by sketching general ideas and this descriptions were not widely understood. But recent works [20, 21] have undertaken to clarify and extend the basic principles. Many additional features to intensify and to diversify the search are proposed and added into the original tabu search algorithm. Simultaneously, many additional parameters are required. Of course, we could obtain an algorithm which works faster and gives better solutions by incorporating many additional *ad hoc* rules; but inserting many features makes the comparison rather vague. So we decided to make the algorithm as simple as possible.

We had revised and tailored the tabu search algorithm in various ways to a number of different problems, and each time learned something we felt was useful. Finally we obtained a version that we found to be quite effective, and that we believed to contain aspects that were not an explicit part of customary tabu search implementations. We developed a mathematical formulation in terms of symmetric differences that has proved for us to be highly convenient, and that has led us to develop our refinements more readily. This has also led us to adopt a philosophy that differs from the philosophy of many tabu search implementations, which are based on collecting principles of intelligent problem solving [21], and which can sometimes require the use of many control parameters. Meanwhile, our philosophy is to keep the number of control parameters as small as possible. We felt that the final version was not tabu search any more; so we choose to call the "life span method" and to clarify these ideas with the goal of allowing them to be implemented more routinely and conveniently.

3.2. Outline of the life span method

Now, we present the LSM in a general form. Note that the LSM is a convenient and useful variant of the original tabu search. We can see the similar type of organizing memory in recent works. The LSM works on 2^B instead of X, where B is the ground set. So the solutions which are not in the feasible solution set X are allowed to be searched. Although some tabu search and simulated annealing algorithms in the literature have adopted such infeasible solution approach, the LSM treats the infeasibility of solutions in an explicit manner.

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 $X \subseteq 2^B$; so 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 'attribute' mapping ψ is simply stated as $\psi(x, x') = x \triangle x'$ in the LSM. For each element β of B, we define 'Life Span' of β as the remaining iterations that β is

forbidden to be used, 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 $\beta \in x \triangle x'$. For every iteration, we decrease $LS(\beta)$ by 1 if $LS(\beta) > 0$. If $LS(\beta)$ is positive, all moves (x, x') whose symmetric differences include β are forbidden. We can say that symmetric difference is a convenient way to summarize the identity of attributes (variables) that change in moving between solutions. Glover emphasized this identification in [21]. The early tabu search literature suggests the merit of asymmetric tabu tenures, applied to solution elements that change from iteration to iteration (hence applied to the members of our symmetric difference set). However, we focus on using completely symmetric tabu tenures, which causes a given element to remain tabu for the same length of time regardless of whether it is removed from or added to a previous solution (to create the current solution). By treating all elements of the symmetric difference set in this way, we simplify the range of options to consider. Note that we may use a simple attribute mapping $\psi(x, x') = x' \setminus x$ instead of $\psi(x, x') = x \triangle x'$. This can be seen as a special case of the original definition of the LSM. Alternately, it is a special case of using asymmetric tabu tenures. When a solution x is moved to a new one $x' \in N(x)$, we set $LS(\beta)$ to *tabulength* for $\beta \in (x' \setminus x)$ and to 0 for $\beta \in (x \setminus x')$. Note that our organization of memory relative to the ground state is equivalent to an implicit zero-one representation of attributes and this is common in many applications of tabu search. Nevertheless, our particular way of defining tabu status corresponds to a different choice than often occurs in the literature. That is, in the terminology of [21], we define tabu status in terms of the 'to' attributes of a move (those that belong to the new solution) rather than in terms of the 'from' attributes (those that belong to the original solution). When only one of these sets of elements is considered, in many cases researchers have focused on defining tabu status relative to the 'from' set rather than the 'to' set, in contrast to our approach. We apply this simple variant of the LSM to the maximum stable set problems, the traveling salesman, quadratic assignment, job-shop scheduling problems in Sections 4, 5, and 9, respectively.

As in tabu search, we move to the best neighbor which is not forbidden to be used. Since we are allowed to visit 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 the 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 (LSM)
1 $x :=$ some initial solution
2 $LS(\beta) := 0$ for all $\beta \in B$
3 $tabulength :=$ a positive integer
4 **while** stopping-criterion $\neq$ yes and $best(x) \neq \emptyset$ **do**
5 $\quad x' := best(x)$
6 $\quad LS(\beta) := tabulength$ for all $\beta \in x \triangle x'$
7 $\quad x := x'$

8 $\quad LS(\beta) := LS(\beta) - 1$ for all $\beta \in B$ such that $LS(\beta) > 0$
9 **return** x

Usually, we can execute the operation in line 8 in the course of finding the best neighbor in line 5; so the time complexity of the above algorithm depends on finding the best move (line 5). If $|B|$ is too large, we may store the number of the iteration in $LS(\beta)$ in line 6 and omit the operation in line 8. Then, if the current iteration number is less than $LS(\beta) + \textit{tabulength}$, all moves which use β are forbidden. Note that a similar technique was used in some special applications such as the permutation problem [21], the quadratic assignment problem [41], and the job shop scheduling problem [42].

We then summarize the merits of the LSM. Firstly, we can determine the attributes without any ambiguity. Further checking the tabu status can be done in $O(1)$ time in the LSM, while the queue implementation of tabu search requires $O(\textit{tabulength})$ time to do the same operation. Instead, the LSM requires an additional $O(|B|)$ memory, but it can be negligible in almost all applications. We can easily incorporate 'randomness' into the algorithm by randomizing the parameter, *tabulength*. Further, allowing the infeasible solutions makes it possible to search much deeper neighborhood. To ensure the final solution to be feasible, some techniques are needed. Examples are the penalty function approach, the fixed cardinality approach, and the pseudo-feasible search, which will be discussed below. Of course, we can add any feature used in tabu search such as the long-term memory and the target analysis to diversify the search into the LSM.

3.3. How to keep feasibility

One characteristic of the LSM is to allow an infeasible solution temporally. In order to keep the feasibility, we need some mechanism to force the solutions to return to the feasible area. We outline three techniques, the penalty function approach, the fixed-cardinality approach, and the pseudo-feasible solution approach. These approaches are not new; they have been used in some local search techniques. But we feel that it is of use to summarize these approaches in an explicit manner here. We also summarize the merits and the defects of these three approaches.

We turn to the analogy again. Finding the best feasible solution using neighborhood structures is similar to the man finding the goal in the maze. Feasible regions of combinatorial optimization problems correspond to roads in the maze. Searching the goal in the maze is too difficult if the man walks on the roads in the maze (this corresponds to the 'usual' feasible solution approach). Sometimes there may not a path from the position where he stands to the goal. The infeasible solution approach corresponds to the man who has wings. He can fly and get over the walls in the maze. He has more freedom than the man on the ground.

3.3.1. Penalty function approach

For penalty function approaches, we represent the infeasibility by the gap between the set of feasible solutions X and an infeasible solution x. Such a gap is usually defined as the distance function $d(x, X)$ between x and X, Of course, we may use any metric function to represent a gap between x and X. We then define the penalty function $p(x)$ as follows:

$$p(x) = c(x) + \alpha \cdot d(x, X), \tag{1}$$

where α is a scalar parameter. The parameter α is determined so that it satisfies the following condition:

$$\frac{c(y) - c(x)}{d(x, X)} < \alpha \qquad \text{for all } y \in X, x \in 2^B \setminus X.$$

Otherwise, an infeasible solution may be an optimal solution with respect to the modified cost function defined in (1). Note that simulated annealing algorithms to the graph coloring problem using the penalty function approach [1, 28] do not satisfy the above condition, but it works well because the algorithm searches locally optimal solutions which are very good in terms of the original cost function.

By adding penalty terms, the algorithm based on neighborhood structures can escape from locally optimal solutions. Meanwhile, it is sometimes difficult to set parameter α appropriately and to select a good penalty function.

3.3.2. Fixed cardinality approach

In some applications, we can guess the objective function value and the aim is to find a solution which attains such an objective function value. In such cases, we fix the objective function value temporally, and decrease the infeasibility until the solution becomes feasible. We call this approach the *fixed cardinality approach*.

For example, consider the 'standard' graph coloring problem (we will discuss a 'fixed cardinality' variant in Section 8): given a graph, find a minimum number of colors so that the color on the pair of vertices incident to any edge is different. For random graphs, we can easily guess the minimum number of colors needed to color the graph. We fix the number of colors temporally, and then solve the (fixed cardinality) problem to minimize the number of edges whose both end vertices have the same color. If a solution with 'zero' value is found, we get an approximate solution to the original graph coloring problem.

If the optimal objective function value is not known *a priori*, we can apply the fixed cardinality approach with a binary search technique. If we know that the (integer) optimal value is in the range $[L, U]$, we can obtain the optimal solution in $\log(U - L)$ time.

The fixed cardinality approach is useful when we can estimate the optimal value *a priori*. Meanwhile, it takes much computational time when the estimation is poor.

3.3.3. Pseudo-feasible solution approach

Let $\tilde{X} \subseteq 2^B$ be the set of pseudo-feasible solutions which is, in a sense, 'near' to the set X of feasible solutions. We search good solutions on $\tilde{X}$ instead of X or 2^B. The precise meaning of the nearness depends on the neighborhood structure. Usually, we use two neighborhoods; one is to escape from the feasible region and the other is to return to the feasible region. A pseudo-feasible solution is an infeasible solution which can be reached from a feasible solution using only one 'escaping' neighborhood.

Note that the similar mechanism has been incorporated in the classical local search algorithms. Two famous classical approaches, Kernighan and Lin's algorithm for the graph partitioning problem [29] and Lin and Kernighan-opt for the traveling salesman problem [33], used the depth-first local search in which the concept of 'pseudo-feasible solution' was used in an implicit manner.

3.4. Overview of Applications

In the following sections, we illustrate the LSM using several combinatorial optimization problems all of which are known to be $\mathcal{NP}$-hard. In Section 4, we consider a simple graph theoretic combinatorial optimization problem, the maximum stable set problem, to illustrate the application of the LSM. In Sections 5 and 6, we apply the LSM to two famous combinatorial optimization problems, the traveling salesman problem and the quadratic assignment problem, both of which seek to a good permutation of n elements. In Sections 7 and 8, we describe applications of the LSM to two graph theoretic combinatorial optimization problems, the graph partitioning and coloring problems, which were used in our first numerical comparisons. Finally, we illustrate a nonstandard application of the LSM to a more complicated combinatorial optimization problem, the job shop scheduling problem in Section 9.

4. Application to Stable Set Problem

The problems that we will consider in this section are described as follows.

Definition 3 (Maximum Stable Set Problem: MSSP)
Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. A *stable set* of G is a subset of V such that no two vertices of the subset are pairwise adjacent. The *Maximum Stable Set Problem* (MSSP) is to find a stable set of maximum cardinality in G.

Definition 4 (Maximum Clique Problem)
Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. A *clique* is a subset of V such that all the vertices are pairwise adjacent. The maximum clique problem is to find a clique of maximum cardinality in G.

Definition 5 (Minimum Vertex Cover Problem)
Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. A *vertex cover* S is a subset of V such that every edge $(i, j) \in E$ is incident to at least one vertex in S. The minimum vertex cover problem is to find a vertex cover of minimum cardinality in G.

These problems have applications in a wide variety of fields such as project selection, classification theory, fault tolerance, coding theory, computer vision, economics, information retrieval, signal transmission theory, aligning DNA and protein sequences, etc. (see [40]).

The *complement* of $G = (V, E)$ is a graph $\bar{G} = (V, \bar{E})$ such that $(i, j) \in \bar{E}$ if and only if $(i, j) \notin E$. It is easily seen that S is a stable set of G if and only if S is a clique of $\bar{G}$ and $V \setminus S$ is a vertex cover of G; thus the maximum clique problem, the vertex cover problem and the maximum stable set problem are equivalent.

In this section, we describe the LSM for solving the MSSP [14]. Note that we can easily construct an algorithm for the maximum clique problem, the minimum cover problem, and a weighted version of these problems based on the algorithm presented below. An implementation of tabu search for the MSSP was presented by Friden et al. [11] and by Gendreau et al. [17], but our implementation based on the LSM is much simpler.

Before describing the details, we reformulate the MSSP as a general combinatorial optimization problem. For the MSSP, the ground set B is the vertex set V. Given a set of vertices $S \subseteq V$, let us denote by $E(S)$ the set of edges whose endpoints are both in S. Then the set X of feasible solutions is defined by

$$X = \{S \subseteq V : |E(S)| = 0\}.$$

Since we want to maximize the cardinality of the stable set S, the cost mapping c is defined by

$$c(S) = -|S|.$$

To design an efficient LSM tailored to the MSSP, we must determine several features of the algorithm very carefully. The main features are:

1. definition of the search space (fixed cardinality approach or penalty function approach or pseudo-feasible search);
2. selection of the neighborhood;
3. selection of life span;
4. how to compute the change in costs.

We will describe the details below.

We adopted the 'pseudo-feasible' solution approach instead of using the fixed-k or penalty function approach. We use a 'move' neighborhood which consists of 'add' and 'drop' phases. Given a vertex set $S(\subseteq V)$, the add and drop neighborhoods are defined by

$$\mathcal{N}_{add}(S) = \{S \cup \{i\} : i \in V \setminus S\} \tag{2}$$

and

$$\mathcal{N}_{drop}(S) = \{S \setminus \{i\} : i \in S\}, \tag{3}$$

respectively.

If $|E(S)| = 0$, we use the add operation; otherwise, we use the drop operation.

For the add neighbor $\mathcal{N}_{add}$, the symmetric difference $x \triangle x'$ of two solutions x and x' such that $x' \in \mathcal{N}_{add}(x)$ is the added vertex i in (2). Similarly, for the drop neighbor $\mathcal{N}_{drop}$, the symmetric difference is the dropped vertex i in (3). Associated with each vertex $i \in V$, we keep the life span $LS(i)$ in which we store the remaining iterations that vertex i is forbidden to be used. If $LS(i) = 0$, vertex i can be added or dropped.

To design a fast algorithm, we must compute the change in costs efficiently. This can be achieved by introducing an auxiliary array δ. For each $i \in V$, $\delta(i)$ keeps the number of vertices $j \in S$ adjacent to i. The array δ can be updated in $O(|V|)$ time using the algorithm presented below.

Now, we can describe the outline of the life span method for the MSSP.

The computational requirement of the algorithm above is $O(|V|)$ per iteration.

```
procedure LSM for MSSP
 1 S = ∅
 2 δ(i) = 0 for all i ∈ V
 3 z := 0   /* z keeps |E(S)| */
 4 LS(i) := 0 for all i ∈ V
 5 while  terminate-criterion ≠ yes do
 6     if z = 0 then     /* add phase */
 7         i* := arg min{δ(i) : i ∈ V \ S, LS(i) = 0}
 8         S := S ∪ {i*}
 9         LS(i*) := tabulength
10         for all  j adjacent to i*
11             δ(j) := δ(j) + 1
12             if  j ∈ S then  z := z + 1
13     else   /* drop phase */
14         i* := arg max{δ(i) : i ∈ S, LS(i) = 0}
15         S := S \ {i*}
16         LS(i*) := tabulength
17         for all  j adjacent to i*
18             δ(j) := δ(j) − 1
19             if  j ∈ S then  z := z − 1
20     for all  i ∈ V
21         if  LS(i) > 0 then  LS(i) := LS(i) − 1
22 return S
```

5. Application to Traveling Salesman Problem

Next, we illustrate the LSM using the most famous combinatorial optimization problem, the traveling salesman problem.

Definition 6 (Traveling Salesman Problem: TSP)
Given a set $V = \{1, \cdots, n\}$ and an $n \times n$ symmetric matrix $D = (d_{ij})$, find a cyclic permutation $\rho : V \to \{1 \cdots, n\}$ which minimizes the cost function

$$c(\rho) = \sum_{i=1}^{n-1} d_{\rho(i)\rho(i+1)} + d_{\rho(n)\rho(1)}.$$

This problem has the following interpretation. A salesman wants to visit n cities, and the distance d_{ij} is the inter-travel distance between cities i and j. The i-th element of the cyclic permutation ρ represents the i-th visiting city of the salesman; the cost $c(\rho)$ is the total travel distance of the salesman. The objective of the TSP is to find a tour of the salesman which minimizes the total travel distance.

Using 0-1 variable x_{ij} which is set to 1 if the salesman visits city j immediately after city i, the TSP is stated as the following integer programming problem:

$$\min \sum_i \sum_j c_{ij} x_{ij}$$

subject to

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

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

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq \mid S \mid -1 \quad S \subset V, S \neq \emptyset,$$

$$x_{ij} \in \{0, 1\} \quad (i, j) \in E.$$

For the TSP, the ground set B corresponds to the edge set E. A feasible solution $x \in X$ for the TSP corresponds to a tour or a cyclic permutation. We denote the set of all cyclic permutations by P and denote the edge set of the tour associated with a cyclic permutation ρ by $T(\rho)$. We say two tours are k-opt neighbors if one can be obtained from the other by deleting k edges and by adding k edges. The most famous local search algorithms are 2-opt and 3-opt procedures introduced by Lin [32]. We define these two neighborhood structures more precisely.

Definition 7 (2-opt neighborhood for TSP)
The 2-opt neighborhood is

$$\begin{aligned} N_2(\rho) = \{\rho' \in P : T(\rho') = T(\rho) \setminus \{e_1, e_2\} \cup \{e_3, e_4\} \\ \text{for all } e_1, e_2 \in T(\rho), e_3, e_4 \notin T(\rho)\}. \end{aligned} \tag{4}$$

Definition 8 (3-opt neighborhood for TSP)
The 3-opt neighborhood is

$$\begin{aligned} N_3(\rho) = \{\rho' \in P : T(\rho') = T(\rho) \setminus \{e_1, e_2, e_3\} \cup \{e_4, e_5, e_6\} \\ \text{for all } e_1, e_2, e_3 \in T(\rho), e_4, e_5, e_6 \notin T(\rho)\}. \end{aligned} \tag{5}$$

A more sophisticated neighborhood is presented by Lin and Kernighan [33]. The Lin and Kernighan-opt neighborhood informally defined as

$$\begin{aligned} N_{LK}(\rho) = \{\rho' \in P : T(\rho') \text{ is obtained from } T(\rho) \\ \text{by successive deletion and addition of edges } \}. \end{aligned}$$

The precise definition of the Lin and Kernighan-opt neighborhood is too complicated to be described here in detail. We refer the readers to [33].

We can easily see that the symmetric difference of two feasible solutions, which are neighbors each other in terms of the above neighborhoods, is the set of deleted and added edges. Notice that the ground set of the TSP is the set E of edges, and that we define the life span $LS(e)$ for each edge $e \in E$. We set $LS(e)$ to a positive number when edge e is deleted from the current tour. If $LS(e)$ is positive, edge e is forbidden to be added again.

In tabu search algorithms, the definitions of attributes were rather vague. Malek et al. [34] proposed a tabu search algorithm for the TSP based on the 2-opt neighborhood in which 9 types of attributes are used. Glover [18] suggested a tabu search algorithm in which the attribute is a pair of edges for the 2-opt neighborhood and a triple of edges for the 3-opt neighborhood. The life span LS can be seen as a projection of the attribute space onto the edge space. Based on the LSM, we can easily construct an algorithm based on the Lin and Kernighan-opt neighborhood.

Using a neighborhood mapping N, which corresponds to N_k for $k = 2$ or 3 or N_{LK}, we can obtain the LSM for the TSP.

procedure LSM for TSP
1 select $\rho(\in P)$ arbitrary
2 $LS(e) := 0$ for all $e \in E$
3 **while** terminate-criterion $\neq$ yes **do**
4 $\quad\rho^* := \arg\min\{c(\rho') : \rho' \in N(\rho), LS(e) = 0 \text{ for all } e \in T(\rho') \setminus T(\rho)\}$
5 $\quad LS(e) := \textit{tabulength}$ for all $e \in T(\rho) \setminus T(\rho^*)$
6 $\quad\rho := \rho^*$
7 $\quad$**for all** $i \in E$
8 $\quad\quad$**if** $LS(e) > 0$ **then** $LS(e) := LS(e) - 1$
9 **return** ρ

In line 4 of the above algorithm, $T(\rho') \setminus T(\rho)$ represents the set of added edges; for the 2-opt neighborhood, it represents $\{e_3, e_4\}$ in (4), while for the 3-opt neighborhood, it represent $\{e_4, e_5, e_6\}$ in (5). Similarly, in line 5, $T(\rho) \setminus T(\rho^*)$ represents the set of deleted edges.

6. Application to Quadratic Assignment Problem

In this section, we apply the LSM to one of the most infamous combinatorial optimization problem, the quadratic assignment problem.

Definition 9 (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 applications of the QAP include the location of machines, ordering of data on a disk, the location of departments (or offices), etc. The QAP is known to be one of the most intractable 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 the object i is assigned to the location j, the QAP is stated as the following integer programming problem:

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

subject to

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

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

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

The ground set B of the QAP is $V \times V$. 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 set of all permutations by Π.

Definition 10 (2-opt neighborhood for QAP)
Given a permutation π, a 2-opt neighbor N is defined by

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

Definition 11 (3-opt neighborhood for QAP)
Given a permutation π, a 3-opt neighbor N is defined by

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

The life span LS is defined on the ground set $B = V \times V$. The symmetric difference of two permutations is a set of pairs of object i and location k. For each pair $(i,k) \in V \times V$, we keep the life span $LS(i,k)$, which is set to a positive value when object i is moved from location k. In the following *tabulength* iterations, object i is forbidden to return to location k again. We can use the same mechanism for the 3-opt neighborhood. We denote the permutation matrix ($n \times n$ square 0-1 matrix whose row and column sums are all 1) 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 $M_{\pi'}(i,j) = 0$. When we move from

π to π', we set $LS(i,k)$ to a positive integer, $tabulength$, for all $(i,k) \in \mathcal{M}(\pi \setminus \pi')$. The life span LS is diminished by 1 for each iteration, and object i is forbidden to move to location k if $LS(i,k)$ is positive.

To implement an efficient local search algorithm, we must compute the difference $c(\pi') - c(\pi)$ in costs of two permutations π and π'.

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

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

We select the best move which minimizes Δ_{ij} such that $LS(i,\pi(j)) = 0$ and $LS(j,\pi(i)) = 0$.

A straightforward implementation takes $O(n^3)$ time per iteration. If we choose the best one among $O(n^2)$ candidate pairs, 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:

$$\Delta_{ij} = \delta_{j\pi(i)} - \delta_{j\pi(j)} + \delta_{i\pi(j)} - \delta_{i\pi(i)} + 2 f_{ij} d_{\pi(i)\pi(j)}. \tag{6}$$

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.

Similarly, we can compute the difference Δ_{ijh} in costs when we exchange three objects i, j, h as follows:

$$\begin{aligned}
\Delta_{ijh} = \; & \delta_{i\pi(j)} - \delta_{i\pi(i)} + \delta_{j\pi(h)} - \delta_{j\pi(j)} + \delta_{h\pi(i)} - \delta_{h\pi(h)} \\
& + f_{ih}(d_{\pi(j)\pi(i)} - d_{\pi(j)\pi(h)}) \\
& + f_{ji}(d_{\pi(h)\pi(j)} - d_{\pi(h)\pi(i)}) \\
& + f_{hj}(d_{\pi(i)\pi(h)} - d_{\pi(i)\pi(j)}) \\
& + f_{ij} d_{\pi(i)\pi(j)} + f_{jh} d_{\pi(j)\pi(h)} + f_{ih} d_{\pi(i)\pi(h)}.
\end{aligned} \tag{7}$$

If three objects a, b, and c exchange 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.

Using the neighborhood mapping N, which is 2 and/or 3-opt neighborhoods, we describe the LSM for the QAP.

procedure life span method for QAP.

```
 1 select π(∈ Π) arbitrary
 2 LS(i, j) := 0 for all (i, j) ∈ V × V
 3 while  terminate-criterion ≠ yes do
 4     while change23-criterion ≠ yes do /* 2-opt phase */
 5         π* := arg min{c(π') : π' ∈ N_2(π), LS(i, k) = 0 for all (i, k) ∈ M(π' \ π)}
 6         LS(i, k) := tabulength for all (i, k) ∈ M(π \ π*)
 7         π := π*
 8         for all  (i, j) ∈ V × V
 9             if  LS(i, j) > 0 then  LS(i, j) := LS(i, j) − 1
10     while change32-criterion ≠ yes do /* 2 and 3-opt phase */
11         π* := arg min{c(π') : π' ∈ N_3(π) ∪ N_2(π), LS(i, k) = 0 for all (i, k) ∈ M(π' \ π)}
12         LS(i, k) := tabulength for all (i, k) ∈ M(π \ π*)
13         π := π*
14         for all  (i, j) ∈ V × V
15             if  LS(i, j) > 0 then  LS(i, j) := LS(i, j) − 1
16 return π
```

7. Application to Graph Partitioning Problem

In this section we focus on the graph partitioning problem which has applications in circuit board wiring and program segmentation.

Definition 12 (Graph Partitioning Problem: GPP)
We are given an undirected graph $G = (V, E)$ with the set of vertices V and the set of edges E. Associated with an edge $(i, j) \in E$, there exists a nonnegative weight c_{ij} called the *cost* of (i, j). Assume that $|V|$ is even and let $n = |V|$. A partition of V is a pair (L, R) of vertex sets such that $L \cap R = \emptyset$ and $L \cup R = V$. Sets L and R are called left and right vertex sets, respectively. A partition (L, R) of V is called *uniform* when $|L| = |R| = n/2$. The objective is to find a uniform partition (L, R) of V which minimizes $c(L, R) = \sum_{i \in L, j \in R} c_{ij}$.

We restrict our attention to the case $c_{ij} = 0, 1$, i.e., we set $c_{ij} = 1$ if and only if $(i, j) \in E$. Notice that the problem under this restriction remains $\mathcal{NP}$-hard [16].

We apply the pseudo-feasible solution approach. We use the following neighborhood structure. Given a partition (L, R), which is not necessary uniform, the left-to-right neighborhood $\overrightarrow{N}$ and the right-to-left neighborhood $\overleftarrow{N}$ are defined by

$$\overrightarrow{N}((L, R)) = \{(L', R') : L' = L \setminus \{\ell\}, R' = R \cup \{\ell\} \text{ for all } \ell \in L\},$$

and

$$\overleftarrow{N}((L, R)) = \{(L', R') : L' = L \cup \{r\}, R' = R \setminus \{r\} \text{ for all } r \in R\},$$

respectively.

```
procedure LSM for GPP
 1 (L, R) := some uniform partition
 2 LS(i) := 0 for all i ∈ V
 3 c* := Σ_{i∈L,j∈R} c_ij
 4 compute S(i) and D(i)
 5 while  stopping-criterion ≠ yes do
 6     ℓ* := arg min{δ(ℓ) : ℓ ∈ L, LS(ℓ) = 0}    /* left-to-right movement */
 7     LS(ℓ*) := tabulength
 8     update S and D
 9     r* := arg min{δ(r) : r ∈ R, LS(r) = 0}    /* right-to-left movement */
10     LS(r*) := tabulength
11     update S and D
12     (L, R) := (L \ {ℓ} ∪ {r}, R \ {r} ∪ {ℓ})
13     for all  i ∈ V
14         if  LS(i) > 0 then  LS(i) := LS(i) − 1
15 return (L, R)
```

The symmetric difference of two solutions is the vertex moved to a different set. Since the ground set B of the GPP is the set V of vertices, we keep $LS(i)$ for each vertex $i \in V$ in which the remaining iterations that vertex i is forbidden to be moved. We set $LS(i)$ to a positive value when vertex i is moved to another set, and decrease $LS(i)$ by 1 for each iteration.

We compute the reduction (or gain) of the cost $c(L', R') - c(L, R)$ using two auxiliary arrays S and D. The arrays S and D are n-dimensional arrays. The i-th element of S (or D) keeps the number of edges incident to vertex i whose endpoints are both in the same set (or the different sets) of a partition. The reduction of the cost $\delta(i)$ by moving vertex i from one set to another may be computed as

$$\delta(i) = S(i) - D(i). \tag{8}$$

Initially the arrays S and D are computed in $O(n^2)$ time. When we move vertex ℓ from L to R, we change the arrays $S(i)$ and $D(i)$ for every vertex i incident to ℓ as follows: $S(i) := S(i) - 1, D(i) := D(i) + 1$ for all $i \in L \setminus \{\ell\}$ incident to vertex ℓ, and $S(i) := S(i) + 1, D(i) := D(i) - 1$ for all $i \in R$. This takes only $O(n)$ time. For the reverse movement, we can update the arrays S and D in a similar manner in $O(n)$ time. Thus, one iteration of the movement can be done in $O(n)$ time.

Using the above strategies as ingredients, we can describe the LSM for the GPP in a simple form. Although we also use the two-move neighbor (we move two vertices from L to R and then return two other vertices from R to L) to accelerate the search in our real code, here we omit the details for simplicity of the description.

8. Application to Graph Coloring Problem

Here we describe a nonstandard application of the LSM to the Graph Coloring Problem (GCP). The GCP has applications to time tabling or scheduling, tool allocation in FMS, frequency assignment, register allocation, and printed circuit board testing.

Definition 13 (Graph Coloring Problem: GCP)
We are given a fixed positive number k and an undirected graph $G = (V, E)$. We want to find a partition of V into k color classes $C_1, \cdots, C_k$ which minimizes the total number of edges which do not have endpoints in different color classes.

For the GCP, the ground set B corresponds to the set V of vertices. In this case, a feasible solution is not a subset of $B(= V)$, but an assignment of k colors to the vertex set V, i.e., the set X of feasible solutions is a subset of K^B, where $K = \{1, \cdots, k\}$. The cost mapping c is defined as the total number of bad edges, where a bad edge is an edge whose both end vertices have the same color. Let $baddegree(i)$ be the number of bad edges incident to vertex i. we restrict the neighborhood moving vertices which have positive $baddegree$.

Now, we are given a partition $\Upsilon = \{V_1, \cdots, V_k\}$ of V into k color classes. We denote by $color(i)$ the index of color class to which vertex i is assigned, i.e., $\ell = color(i)$ if and only if $i \in V_\ell$. The neighborhood N of $\Upsilon = \{V_1, \cdots, V_k\}$ for the GCP is defined by

$$N(\Upsilon) = \{\Upsilon' = \{V_1', \cdots, V_k'\} : V'_{color(i)} = V_{color(i)} \setminus \{i\}, V'_\ell = V_\ell \cup \{i\}$$
$$\text{for all } \ell \neq color(i) \text{ and } i \in V \text{ such that } baddegree(i) > 0\}.$$

We select the best neighbor as follows. We first compute the number $C(i, \ell)$ of bad edges incident to vertex i when i is assigned to the color class $\ell \neq color(i)$. In order to compute $C(i, \ell)$ efficiently, we must carefully choose the data structure. The underlying graph $G = (V, E)$ is represented by an adjacency list consisting of an array of $|V|$ lists and pointers to all adjacent vertices. We use a table T with length k. For each vertex i which has a positive $baddegree$, we execute the following procedure.

subroutine computation of $C(i, \ell)$
1 $T(\ell) := 0$ for all $\ell = 1, \cdots, k$
2 **for all** j adjacent to vertex i
3 $\quad T(color(j)) := T(color(j)) + 1$
4 **return** T

The above subroutine returns an array T whose ℓ-th element corresponds to $C(i, \ell)$. Thus, we can compute all $C(i, \ell)$'s in $O(n^2)$ time. Let $k^*(i) \neq color(i)$ be the color class which attains the minimum of $C(i, \ell)$ over all $\ell \in K \setminus \{color(i)\}$. We select the best vertex among the candidates as follows:

$$i^* = \arg\min\{C(i, k^*(i)) : i \in V, LS(i) = 0, baddegree(i) > 0\}.$$

Then we set the color class of vertex i to $k^*(i)$ and set $baddegree(i^*)$ to $k^*(i^*)$; this can be done in $O(1)$ time.

Associated with each vertex $i \in V$, we define the life span $LS(i)$. We set $LS(i)$ to a positive integer, $tabulength$, when the color of vertex i is changed. The life span of vertex i is the remaining number of iterations that vertex i is forbidden to be used as a candidate vertex.

```
procedure LSM for GCP
 1 color(i) := a random integer from {1, ..., k} for all i ∈ V
 2 LS(i) := 0 for all i ∈ V
 3 compute baddegree(i) for all i ∈ V
 4 while stopping-criterion ≠ yes do
 5     Θ := {i ∈ V : baddegree(i) > 0 and LS(i) = 0}
 6     compute C(i, ℓ) for all ℓ ≠ color(i), LS(ℓ) = 0, and i ∈ Θ
 7     k*(i) := arg min{C(i, ℓ) : ℓ ∈ K} for all i ∈ Θ}
 8     i* := arg min{C(i, k*(i)) − baddegree(i) : i ∈ Θ}
 9     color(i*) := k*(i*)
10     LS(i*) := tabulength
11     for all i ∈ V
12         if LS(i) > 0 then LS(i) := LS(i) − 1
13     for all ℓ ∈ K
14         if LS(ℓ) > 0 then LS(ℓ) := LS(ℓ) − 1
15 return color
```

9. Application to Job Shop Scheduling Problem

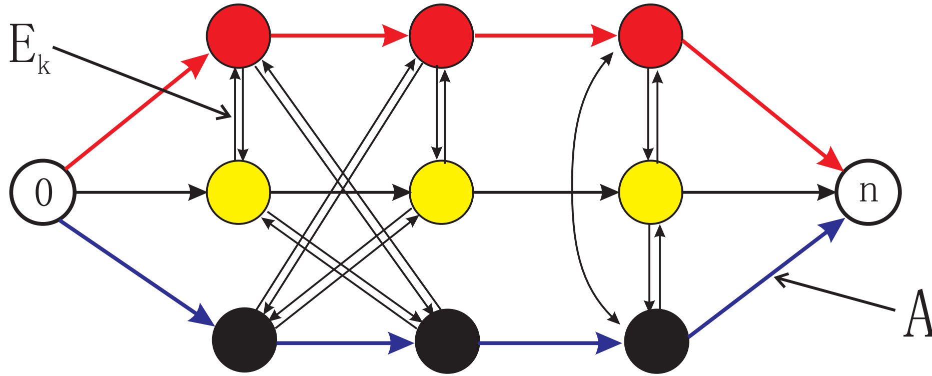

Figure 1: The disjunctive graph

In this section, we describe our last application of the LSM to the job shop scheduling problem which has been known as a notoriously difficult combinatorial optimization problem.

Definition 14 (Job Shop Scheduling Problem: JSSP)
Let $N = \{0, 1, \cdots, n\}$ the set of operations which must be processed on the set $M = \{1, \cdots, m\}$ of machines where 0 and n denote the starting and finishing dummy operations, respectively. Let A be the set of pairs of operations corresponding to precedence (linear ordering) relations between operations. For each machine $k \in M$, we denote by E_k the set

of pairs of operations performed on machine k. The objective is to find a selection S_k in E_k which contains exactly one of the pairs in E_k so that the completion time of the finishing operation n is minimized. JSSP is often described by a disjunctive graph $G = (V, A, E)$. Figure 1 shows the disjunctive graph G. Using graph theoretic terms, we are given a set of nodes N, the set of *conjunctive arcs* A, and the set of *disjunctive arcs* $E = \cup_{k \in M} E_k$. The objective is to select one of disjunctive arcs between two operations so that the length of the longest (critical) path between 0 and n is minimized .

We use the neighborhood used in the simulated annealing algorithm [44] and tabu search [42]. Let $D(p)$ the set of disjunctive arcs on longest path p. Then, the neighborhood N of longest path p is defined by

$$N(p) = \{p' : p' \text{ can be obtained from } p \text{ by reversing a disjunctive arc } (i,j) \in D(p)\}.$$

In this case, the ground set B is the set of disjunctive arcs. So we keep the life span $LS(i,j)$ for each disjunctive arc $(i,j) \in \cup_{k \in M} E_k$. Given an operation i, let $a(i)$ and $b(i)$ be the operations that are processed before and after i, respectively, (if it exists). When we reverse a disjunctive arc (i,j), we set $LS(i,j)$, $LS(u,i)$, and $LS(j,v)$ to a positive number, *tabulength*, where $u = b(i)$ and $v = a(j)$.

The change in costs, i.e., the length of the longest path, can be computed in $O(n)$ time using the (Bellman-Ford) labeling method (see, for example [31, 39]) for each disjunctive arc on the current longest path.

procedure LSM for JSSP
1 $p :=$ a critical path obtained by a list scheduling algorithm
2 $LS(e) := 0$ for all $e \in \cup_{k \in M} E_k$
3 **while** stopping-criterion $\neq$ yes **do**
4   **for all** disjunctive arcs (i,j) on p
5   find a critical path after reversing (i,j)
6   let p be the shortest critical path
7   let (i,j) be the corresponding disjunctive arc
8     $LS(i,j) := LS((b(i),i)) := LS((j,a(j))) := \textit{tabulength}$
9   **for all** $i \in V$
10   **if** $LS(i) > 0$ **then** $LS(i) := LS(i) - 1$
11 **return** p

10. Results of Experiments

We briefly summarize our experimental results.

Table 1: Results on random graphs with $p = 0.5$ for the MSSP.

graph : $G(n,p)$	$\hat{\beta}$	LSM Avg	LSM Max	Friden [12] Avg	Friden [12] Max	Gendreau [17] Avg	Gendreau [17] Max	Feo [9] Ave	Feo [9] Max	Johnson [28] Ave	Johnson [28] Max
$G(100, 0.5)$	9	9	9	9	9	9	9	-	-	8.6	9
$G(300, 0.5)$	12	12	12	12	12	11.5	12	-	-	10.9	12
$G(500, 0.5)$	13	13	13	13	13	12.7	13	-	-	11.8	13
$G(1000, 0.5)$	15	15	15	15	15	-	-	15	15	13.0	15
$G(1500, 0.5)$	16	16	16	15.6	16	-	-	15.9	16	13.7	15
$G(2000, 0.5)$	17	16.9	17	-	-	-	-	16.8	17	14.1	16
$G(4000, 0.5)$	18	17.3	18	-	-	-	-	-	-	15.1	16

	The computational environments
LSM	Hitachi 3050
Friden	VAX Station II/RC
Gendreau	IBM PS/2 MODEL 70
Feo	Alliant FX/80 parallel/vector computer
Johnson	SGI Challenge

Table 2: Results on DIMACS Benchmarks for the MSSP.

File	n	Edges	Clique Size	Time(sec.)
c-fat200-1	200	1534	12	0.01
c-fat200-2	200	3235	24	0.05
c-fat200-5	200	8473	58	0.15
c-fat500-1	500	4459	14	0.05
c-fat500-10	500	46627	126	0.52
c-fat500-2	500	9139	26	0.03
c-fat500-5	500	23191	64	0.23
johnson16-2-4	120	5460	8	0.02
johnson32-2-4	496	107880	16	0.03
johnson8-2-4	28	210	4	0.01
johnson8-4-4	70	1855	14	0.01
keller4	171	9435	11	0.07
keller5	776	225990	27	17.27
keller6	3361	4619898	59	2113.20
hamming10-2	1024	518656	512	7.37
hamming10-4	1024	434176	40	0.56
hamming6-2	64	1824	32	0.03
hamming6-4	64	704	4	0.01
hamming8-2	256	31616	128	0.43
hamming8-4	256	20864	16	0.01

10.1. Maximum Stable Set Problem

In this section, we show the numerical results for the MSSP. For more details, see the companion papers [14].

We tested our algorithm on random graphs. For our random graphs we select the model (see [4, 38]) which consists of graphs in which the edges are chosen independently with probability p. If we define the density of a graph G as the number of edges of $G = (V, E)$ over the number of edges of the complete graph with $|V|$ vertices, then for this class of random graphs the density is very close to p. We set $p = 0.5$ because this class of random graphs are the hardest one and the probabilistic estimation of the upper bound of the optimal solution is tight. We tested three instances for each vertex size. The algorithm has been coded in C-language, and the experiments were executed on SPARC station 2 with 16 MB. Running time were measured by making the system call **times** and converting to seconds.

The efficient heuristics known in the literature are tabu search algorithms due to Friden et al. [12] and Gendreau et al. [17]. Feo et al. [9] proposed the greedy randomized adaptive search procedure (GRASP) for the maximum stable set problem. Dmclique is a variant on the simple 'semi-exhaustive greedy' scheme for finding large stable sets used in the graph coloring algorithm XRLF described in Johnson et al. [28]. We compare the performance of the LSM with their methods.

Table 1 shows the results of experiments on random graphs with $p = 0.5$. Friden et al.[12] sometimes failed to obtain stable sets of size 16 on the instances with $n = 1500$. Our algorithm consistently finds the solutions whose values are equal to the probabilistic estimates when the size n is 1500 or less. Gendreau et al.[17] did not always obtain the stable sets of size 13 on the instances with $n = 500$. We see no major difference between the GRASP and the LSM on random graphs, and the GRASP was competitive with the LSM. Although we consider the computational environments, Dmclique was very fast, but the results for random graphs were inferior to other algorithms.

We also tested our algorithm on DIMACS test problems in anonymous ftp site `dimacs.rutgers.edu`. Though, the test instances are for the maximum clique problem, we can obtain the stable set instances by complementing the edges very easily. In fact, we modify the program by simply adding a "not (!)" clause to the edge macro. The results are shown in Table 2; all clique sizes obtained by our algorithm are equal to the optimal or best known values.

10.2. Traveling Salesman Problem

Lin and Kernighan algorithm [33] for the TSP uses a deep and complicated neighborhood and it is well known that Lin and Kernighan algorithm is superior to other heuristics. We performed preliminary experiments for comparing the LSM based on the 2-opt neighborhood with Lin and Kernighan algorithm. As a conclusion on the preliminary experiments, the LSM for the TSP is not so bad, but Lin and Kernighan is superior to the LSM on the whole. So we hereafter incorporate a deeper neighborhood like Lin and Kernighan-opt neighborhood to our LSM.

Table 3: Performance comparison of best upper bounds (BUB) and the total number of iterations (TNI) for the QAP.

	LSM [15]			*Augmented Par_tabu* [5]	
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

Table 4: The average behavior of the LSM for the QAP.

	Time(sec.)			Solution		
Prob. 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(100.9)	149694
sko100f	5342.6	5493.1(153.1)	5643.6	149036	149093.3(45.9)	149216

10.3. Quadratic Assignment Problem

In this section, we report the results of our computational experiments. For more details, see the companion papers [15]. All computational experiments were executed on SONY NEWS-5000WI with 128 MB 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 [15]. Then, we compare the performance of the LSM with that of tabu search by Chakrapani and Skorin-Kapov [5] that was known to be one of the best heuristic algorithms.

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 3 summarizes the best upper bounds(BUB) and the total number of iterations(TNI) achieved by the LSM and *Augmented Par_tabu* [5]. We consider that one iteration of 3-opt neighborhood corresponds to n times as large as one iteration of 2-opt neighborhood. Asterisk $*$ indicates the BUB when the LSM exceeds the *Augmented Par_tabu*. Although we consider the iterations of 3-opt phase, every TNI of the LSM is less than that of the *Augmented Par_tabu*.

In Table 4, 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.

10.4. Graph Partitioning Problem

In this section, we show the numerical results for the GPP [13]. An implementation of the LSM for GPP was tested on SPARC station 2 with 16 MB. The programs were written in the C-language. Running time were measured by making the system call **times** and converting to seconds.

We use the following standard set of instances given by Johnson et al. [27]:

- Random graph $\mathcal{G}(n,p)$:
  n-vertex graph obtained by setting a pair of vertices to an edge with probability p independently of each other;

- Geometric graph $(U_{n,d})$:
  Let (x_i, y_i) be the x and y coordinates of vertex i uniformly and independently distributed in the unit square $[0,1]^2$. Set an edge between two vertices i and j if and only if $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \le d$.

We tested uniform instances of the size $n = 124, 250, 500, 1000$ with the expected degree $2.5, 5, 10, 20$, and geometric instances of the size $n = 500, 1000$ with the expected degree $5, 10, 20, 40$.

Table 5 summarizes the results on uniform instances. The values in Table 5 show the estimated means for algorithms. The values of local search (Local Opt: 2000 runs), Kernighan-Lin opt [29](K-L: 2000 runs), simulated annealing algorithm (Annealing: 20 runs), and the best value (Best Found) are due to Johnson et al. [27]. The values of tabu search (Tabu:

Table 5: Average Results on 16 Random Graphs for the GPP.

	Expected Average Degree				
n	2.5	5.0	10.0	20.0	Algorithm
124	24.4	78.2	194.9	474.1	Local Opt
	15.4	67.1	183.5	457.5	K-L
	13.5	64.2	179.0	449.9	Annealing
	13	63	178	449	Best Found
	13	63	178	449	LSM
250	58.4	144.2	396.3	876.0	Local Opt
	35.4	123.8	372.4	843.7	K-L
	32.0	116.0	359.9	831.3	Annealing
	29	114	357	828	Best Found
	29	114	357	828	LSM
500	105.2	291.1	706.5	1845.2	Local Opt
	64.2	244.2	655.6	1785.9	K-L
	57.2	223.8	633.7	1752.7	Annealing
	52	219	628	1744	Best Found
	49*	218*	626*	1744	LSM
1000	210.9	591.7	1537.9	3602.5	Local Opt
	125.0	499.7	1432.6	3480.5	K-L
	109.5	460.0	1376.6	3402.6	Annealing
	102	451	1367	3389	Best Found
	96*	446*	1362*	3382*	LSM

Table 6: Best Results on 8 Geometric Graphs for the GPP.

	Expected Average Degree				
n	5.0	10.0	20.0	40.0	Algorithm
500	29.9	70.1	198.1	412.0	Local Opt
	11.4	26.6	178.0	412.0	K-L
	14.9	44.4	198.1	475.9	Annealing
	4	26	178	412	Best Found
	2*	26	178	412	LSM
1000	99.5	211.7	405.1	784.2	Local Opt
	30.3	56.3	224.9	737	K-L
	35.9	92.5	256.9	790.1	Annealing
	3	39	222	737	Best Found
	1*	39	222	737	LSM

Table 7: Average Running Times in Seconds on 16 Random Graphs for the GPP.

	Expected Average Degree				
n	2.5	5.0	10.0	20.0	Algorithm
124	0.1	0.2	0.3	0.8	Local Opt
	0.8	1.0	1.4	2.6	K-L
	85.4	82.8	78.1	104.8	Annealing
	0.9	1.9	1.8	1.5	LSM
250	0.3	0.4	0.8	1.3	Local Opt
	1.5	2.0	2.9	4.6	K-L
	190.6	163.7	186.8	222.3	Annealing
	6.3	9.6	8.6	19.7	LSM
500	0.6	0.9	1.5	3.2	Local Opt
	2.8	3.8	5.7	11.4	K-L
	379.8	308.9	341.5	432.9	Annealing
	51.0	23.0	68.7	54.3	LSM
1000	2.4	3.8	6.9	14.1	Local Opt
	7.0	8.5	14.9	27.5	K-L
	729.9	661.2	734.5	857.7	Annealing
	122.0	164.0	45.0	95.0	LSM

Table 8: Average Running Times in Seconds on 8 Geometric Graphs for the GPP.

	Expected Average Degree				
n	5.0	10.0	20.0	40.0	Algorithm
500	1.0	1.6	3.2	7.2	Local Opt
	3.4	4.8	7.4	11.1	K-L
	293.3	306.3	287.2	209.9	Annealing
	24.0	25.0	14.0	8.0	LSM
1000	2.2	3.7	7.2	18.0	Local Opt
	7.6	11.9	18.9	28.7	K-L
	539.3	563.7	548.7	1038.2	Annealing
	302.0	280.0	37.0	58.0	LSM

10 run) are our original. Table 6 reports the expected best value encountered in 5 runs of Annealing, or a time-equivalent number of runs of K-L or Local Opt, and 10 run of our result of tabu search. Asterisk (*) in Tables shows that the value obtained by tabu search dominates the best known value.

Tables 7 and 8 summarize the average running times of the algorithms. Average computational times of Local Opt, K-L, and Annealing are on VAX 11-750 computers with floating point accelerators and 3 or 4 MB of main memory, and computational times of Tabu are on SPARC station 2 with 16 MB of main memory. As the computational environments are different each other, we cannot compare the computational time of the LSM with that of other algorithms strictly. But the LSM renews the best known solutions of 9 benchmark problems.

10.5. Graph Coloring Problem

In this section, we give the results of our computational experiments. All computational experiments were executed on **SPARC station** 1+ with 16 MB and algorithms were coded in C. The data structure we adopted for representing graphs is the bitmap. This technique was also used in the programs for the maximum clique problem due to Applegate and Johnson which can be obtained from DIMACS (DIscrete MAthematics and theoretical Computer Science: anonymous ftp site `dimacs.rutgers.edu`).

In our experiments, we use the same graphs which have been used by Johnson et al. [28].

- Random Graphs (Johnson et al. and Morgenstern)
  A random graph is defined by two parameters, n and p, where n describes the number of vertices and p the probability that there is an edge between any pair of vertices independently. In this thesis, we denote the random graph with parameter (n, p) by $\mathcal{G}(n, p)$.

- Cooked Graphs (Johnson et al.)
  A cooked graph is a variant of the random graph and defined by two parameters, n and k. In cooked graphs, a chromatic number $\chi(G)$ is guaranteed to be k. In this paper, we denote the cooked graph with parameter (k, n) by $\mathcal{C}(n, k)$. We can make a cooked graph as follows:

  1. Randomly we assign k colors to vertices.
  2. Choose one representative vertex from each color class and make the *clique* (complete graph) among the representative vertices .
  3. For each pair (u, v) of vertices not in the same color class, draw an edge between u and v with probability $k/2(k-1)$.

We compare three fixed-k approaches; fixed-k annealing [28], Friden and de Werra's fixed-k tabu search [24] and our fixed-k tabu search. Johnson et al. improved the annealing for the graph coloring problem which was proposed by Chams et al. [6]. Table 9 - 11 shows the computational results on the random graphs and the cooked graphs. LB means the probabilistic lower bound. The results are mixed and we can not tell our proposed algorithm is superior to other fixed-k approaches,but in comparison with the Friden and de Werra's

Table 9: The computational results on the random graph $\mathcal{G}(n, 0.5)$ for the GCP.

			proposed fixed-k tabu search		fixed-k annealing [28]		fixed-k tabu search [6]	
$\lvert V\rvert = n$	LB	$\tilde{\chi}(G)$	Colors	Time(sec.)	Colors	Time(sec.)	Colors	Time(sec.)
125	16	17	18	1.1	19	6.6	19	7.0
250	27	29	31	6.8	31	111.2	33	33.1
500	46	49	54	95.3	53	2987.3	57	512.6
1000	85	86	93	527.5	97	6913.2	96	2549.0

Table 10: The computational results on the random graph $\mathcal{G}(n, 0.1)$ for the GCP.

			proposed fixed-k tabu search		fixed-k annealing [28]		fixed-k tabu search [6]	
$\lvert V\rvert$	LB	$\tilde{\chi}(G)$	Colors	Time(sec.)	Colors	Time(sec.)	Colors	Time(sec.)
125	5	5	6	0.1	6	0.4	6	1.7
250	7	8	9	0.8	9	5.4	9	9.5
500	11	13	13	46.8	13	475.0	15	24.6
1000	19	21	23	60.0	27	48.0	26	105.1

Table 11: The computational results on the random graph $\mathcal{G}(n, 0.9)$ for the GCP.

			proposed fixed-k tabu search		fixed-k annealing [28]		fixed-k tabu search [6]	
$\lvert V\rvert$	LB	$\tilde{\chi}(G)$	Colors	Time(sec.)	Colors	Time(sec.)	Colors	Time(sec.)
125	40	43	44	14.2	44	45.6	45	22.3
250	70	71	74	23.3	74	7809.8	75	563.7
500	122	128	139	80.3	133	7721.3	143	1293.1
1000	217	226	247	3864.9	283	5187.7	283	1285.9

fixed-k tabu search [24], the proposed algorithm dominates it in the value of colors and the computational time. The fixed-k approaches cannot get a near optimal value and the best known value of $\chi(G)$.

10.6. Job Shop Scheduling Problem

In this section, we report the numerical results for the JSSP [43]. All computational experiments were executed on SPARC station 2 with 16 MB and algorithms were coded in C language. Running time were measured by making the system call **times** and converting to seconds.

In our experiments, we use benchmark problems of the JSSP. We compare our algorithms with the previous algorithms such as M. Dell'Amico and M. Trubian [8] and J. Adams and E. Balas and D. Zawack's shifting bottleneck procedure [2]. M. Dell'Amico and M. Trubian also applied tabu search to the JSSP. Their algorithms are superior to the previous heuristics but have many parameters which are to be optimized.

Table 12: Results on Benchmark problems for the JSSP

problem	$n \times m$	opt. or (best)	DT [8] C_{best}	DT [8] $\overline{C}$	DT [8] $\overline{t}$	ABZ [2] C	ABZ [2] t	LSM C_{best}	LSM $\overline{C}$	LSM $\overline{t}$
MT10 [37]	10×10	930	935	948.4	155.8	930	851	934	939.1	88.3
MT20 [37]	20× 5	1165	1165	1166.8	260.2	1178	80	1165	1167.0	118.3
ABZ5 [2]	10×10	1234	1236	1237.6	139.5	1239	1503	1238	1238.8	51.4
ABZ6 [2]	10×10	943	943	943.8	86.8	943	1101	943	944.2	64.9
ABZ7 [2]	20×15	(665)	667	675.6	320.1	710	1269	667	671.1	850.6
ABZ8 [2]	20×15	(670)	678	684.2	336.1	716	1775	676	682.5	1158.5
ABZ9 [2]	20×15	(686)	692	700.2	320.8	735	1312	686	691.5	1346.0

$\overline{C}$ The average of objective function value

C_{best} The best of objective function value

$\overline{t}$ The average of computational time

The computational environments

LSM SPARC station 2

ABZ SPARC station 2

DT IBM PC 386 (20 MHz)

Table 12 summarizes the results on these algorithms. Our algorithms are competitive with two other algorithms.

The computational time of the LSM is inferior to the tabu search of Dell'Amico et al. But their algorithms have many parameters and it is difficult to optimize all parameters for each problem. For large problems, the shifting bottleneck procedure requires a large amount of computational time compared with two other algorithms.

11. Concluding Remarks

We showed a new variant of local search called the life span method and illustrated this technique by applying to several combinatorial optimization problems. We have already done extensive experiments including parameter optimization; but we shortly summarized the results of experiments in this paper. All experiments are done on the distributed test instances and/or the newly generated random instances. The results are very encouraging; the proposed algorithms dominate the previously proposed algorithms both in speed and accuracy of solutions. We have found that we can tailor the parameters of our method to each individual problem class, and also to each individual member of the class, very effectively. While often the goal is to have a single set of parameters to apply to many different problems from a given class, we point out that there can also be merit in having a method whose parameters can be adjusted for individual problems until very good outcomes are obtained. All the results are included in the companion papers in which we also describe implementation details. This framework gives very good outcomes for the amount of effort needed to create the implementation.

References

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

[2] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling problem. *Management Science*, 34:391–401, 1988.

[3] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[4] B. Bollobas. *Random Graphs*. Academic Press, 1985.

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

[6] M. Chams, A. Hertz, and D. de Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operations Research*, 32:260–266, 1987.

[7] 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.

[8] M. Dell'Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252, 1993.

[9] T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for maximum independent set. *Opns. Res.*, 42(5):860–878, 1994.

[10] M. A. Fleischer and S. H. Jacobson. *Cybernetic Optimization by Simulated Annealing: An Implementation of Parallel Processing Using Probabilistic Feedback Contorol*. Kluwer Academic Publishers, 1996.

[11] C. Friden, A. Hertz, and D. de Werra. Stabulus: A technique for finding stable sets in large graphs with tabu search. *Computing*, 42:35–44, 1989.

[12] C. Friden, A. Hertz, and D. de Werra. An exact algorithm based on tabu search for finding a maximum independent set in graph. *Computers Opns. Res.*, 17(5):375–382, 1990.

[13] K. Fujisawa, M. Kubo, and S. Morito. Experimental analyses of the tabu search for the graph partitioning problem(in Japanese). *The Institute of Electrical Engineers of Japan*, 114(C)-4:430–437, 1994.

[14] K. Fujisawa, S. Morito, and M. Kubo. Experimental analyses of the life span method for the maximum stable set problem. *The Institute of Statistical Mathematics Cooperative Research Report*, 75:135–165, 1995.

[15] K. Fujisawa, S. Morito, and M. Kubo. Experimental analyses of the life span method for the quadratic assignment porblem. *The Institute of Statistical Mathematics Cooperative Research Report*, 75:166–188, 1995.

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

[17] M. Gendreau, P. Soriano, and L. Salvail. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, 41:385–403, 1993.

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

[19] F. Glover. Tabu search II. *ORSA Journal on Computing*, 2:4–32, 1990.

[20] F. Glover. Tabu search: fundamentals and usage. Working paper, University of Colorado, Boulder, 1995.

[21] F. Glover and M. Laguna. Tabu search. *A chapter in Modern Heuristic Techniques for Combinatorial Problems,* C. Reeves, ed., Blackwell Scientific Publishing, pages 70–141, 1993.

[22] F. Glover, E. Taillard, and D. de Werra. A users guide to tabu search. *special issues of the Annals of Operations Research, J.C. Baltzer*, 41:3–28, 1993.

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

[24] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.

[25] J. H. Holland. *Adaptation in Natural and Artificial Systems*. Michigan Press, 1975.

[26] J. J. Hopfield and D. W. Tank. Computing with neural circuits: A model. *Science*, 233:625–633, 1986.

[27] 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*, 37:865–892, 1989.

[28] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, part II, graph coloring and number partitioning. *Operations Research*, 39:378–406, 1991.

[29] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.

[30] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[31] E. L. Lawler. *Combinatorial Optimization :Network and Matroids*. Rinehart and Winston, New York, 1976.

[32] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965.

[33] S. Lin and W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[34] M. Malek, M. Guruswamy, and M. Pandya. Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research*, 21:59–84, 1989.

[35] Z. Michalewicz. *Genetic Algorithm + Data Structure = Evolution Programs(2nd edition)*. Springer Verlag, 1994.

[36] Z. Michalewicz. *Evolutionary Computation and Heuristics*. Kluwer Academic Publishers, 1996.

[37] J. F. Muth and G. L. Thompson. *Industrial Scheduling*. Prentice-Hall, 1963.

[38] E. M. Palmer. *Graphical Evolution*. Wiley, 1985.

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

[40] P. M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4:301–328, 1994.

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

[42] E. Taillard. Parallel taboo search for the job shop scheduling problem. Technical report, Swiss Federal Institute of Technology of Lausanne, 1989.

[43] H. Takayama, M. Kubo, and S. Morito. Scheduling and tabu search. *Communications of the Operations Research Society of Japan*, 40(1):47–54, 1995.

[44] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

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