Experimental Analyses of the Life Span Method for the Maximum Stable Set Problem

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

[†] Department of Architecture and Architectural Systems,
Kyoto University,
Yoshida-Honmati, Sakyo, Kyoto, 606-8501, Japan
[††] Department of Information Engineering and Logistics,
Tokyo University of Mercantile Marine,
Etsujima, Koutou-ku, Tokyo, 135-8533, Japan

Abstract

An efficient algorithm for the approximate solution of the maximum cardinality stable set problem is presented. The algorithm is based on a variant of tabu search which we call the life span method. Numerical experiments on random and benchmark instances show that our algorithm dominates all the algorithms given in the literature both in accuracy of solutions and in speed. We also investigate how to tune up our implementation and to optimize the parameters via extensive numerical experiments.
Key words: maximum stable set problem, maximum clique problem, approximate algorithm, experimental analyses, tabu search, life span method.

1 Introduction

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. A *clique* is a subset of V such that all the vertices are pairwise adjacent. The maximum clique problem (MCP) is to find a clique of maximum cardinality in G. 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 (MVCP) is to find a vertex cover of minimum cardinality in G. 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.

We should distinguish a *maximum* stable set (clique) from a *maximal* stable set (clique). A maximal stable set (clique) is a stable set (clique) that is not a subset of any other stable set (clique). A maximum stable set (clique) is a maximal stable set (clique) that has the maximum cardinality.

The MSSP is known to be $\mathcal{NP}$-hard for arbitrary graphs, which means that unless $\mathcal{P} = \mathcal{NP}$, there exists no algorithm that finds an optimal solution in polynomial time. Furthermore, it can be shown that given an $\epsilon > 0$, there exists no polynomial time algorithm for approximating the maximum clique size within a factor of $|V|^\epsilon$ under the assumption that $\mathcal{P} \neq \mathcal{NP}$ [3].

In this paper, we present an approximate algorithm which is simple and reasonably efficient. The organization of this paper is as follows. In Section 2, we describe the previous work for the MSSP, the maximum clique problem, and the vertex covering problem. In

Section 3, we briefly review the local and tabu search heuristics, and then introduce a variant of tabu search, namely the life span method. In Section 4, we give an application of the life span method to the MSSP. The results of the numerical experiments and parameter optimization are shown in Section 5. The final section gives conclusions.

2 Previous Work

In this section, we review the previous work on the MSSP.

2.1 Complexity

As we have mentioned in Section 1, the maximum stable set (MSSP), the maximum clique (MCP), and the minimum vertex cover (MVCP) problems are computationally equivalent on arbitrary graphs. They are known to be $\mathcal{NP}$-hard.

The new complexity class *MAX SNP* was introduced by Papadimitriou and Yannakakis [32]. They showed that many problems are complete in this class, relative to a reducibility that preserves the quality of approximation. The MAX 3-SAT problem and the vertex cover problem are examples of such complete problems. In [5], Berman and Schnitger have shown that if one of the *MAX SNP* problems does not have a polynomial time approximation scheme, then there is an $\epsilon > 0$ such that the maximum clique cannot be approximated in polynomial time with performance ratio

$$\frac{size\ of\ maximum\ clique}{size\ of\ approximate\ clique} = O(|V|^{\epsilon}).$$

A breakthrough in approximation complexity was made by the recent result of Arora et al. [2, 3]. They showed that the maximum number of satisfiable clauses in a 3-SAT formula (MAX 3-SAT) cannot be approximated to arbitrary small constants (unless $\mathcal{P} = \mathcal{NP}$), thus resolving the open question in [32]. This immediately shows the hardness of finding good approximate solutions to all the above listed problems. In particular, it is shown that no polynomial time algorithm can approximate the maximum clique size within a factor of n^{ϵ} ($\epsilon > 0$), unless $\mathcal{P} = \mathcal{NP}$ (by using the results of Feige et al. [10]).

2.2 Heuristics

The majority of approximation algorithms in the literature for the MSSP, MCP and MCVP. fall into the category known as *sequential greedy heuristics*. These heuristics repeatedly add a vertex into a stable set, or delete of a vertex from a set that in not a stable set to generate a maximal stable set.

Two classes of *sequential greedy heuristics* have been proposed by Kopf and Ruhe [26]. They are composed of the *Best in* and the *Worst out* heuristics. The heuristics decide a vertex to be added in or moved out by referring certain indicators. For example, if the indicator is the degree of a vertex, the *Best in* heuristic adds in a vertex that has the smallest degree among all candidate vertices. The *Worst out* heuristic starts with an initial set V, and repeatedly removes a vertex out of V until V becomes a stable set. All sequential heuristics find only one maximal stable set. The algorithm stops when a maximal stable set is found. We can view these types of heuristics from a different point. For example, Pardalos and Xue

[33] define S_G to be the space consisting of all maximal stable set of G. A sequential greedy heuristic finds one point in S_G.

Given an initial point x, what a *local search heuristic* does, is search its neighborhood and selects the next point x'. One major class of local search heuristics in the literature is the *k-interchange* heuristics. Given a feasible solution x of the MSSP, a k-interchange neighbor $\mathcal{N}_k$ of x is defined by

$$\mathcal{N}_k : x \rightarrow \{y : y \text{ is a stable set }, |x \triangle y| \leq k\},$$

where $x \triangle y$ is the symmetric difference of x and y, i.e., $x \triangle y = (y \backslash x) \cup (x \backslash y)$. Given a feasible stable set x, a k-interchange heuristic searches all the k-neighbors of x and outputs the best (largest) stable set found. It repeats this step until no improved solution can be found. The performance of a local search heuristic depends on the initial solution and the definition of the neighborhood. As the size of the neighborhood increases, the solution quality of a local search improves, but the effort of computation increases rapidly.

A randomized heuristics runs a heuristic (with some random factors included) a number of times to find different points over S_G. For example, Feo et al. [11] proposed an elaborated implementation of the randomized heuristic for the MSSP. Their computational results show that their approach was effective in finding large stable sets on randomly generated graphs.

The simulated annealing algorithm, neural net approach, and tabu search have been used to design heuristics for the MSSP. Aarts and Korst [1] presented an application of the simulated annealing and neural net algorithms to the MSSP. Friden et al. [14] and Gendreau et al. [16] implemented tabu search. Friden et al. used the fixed cardinality approach in which the cardinality of the set S is temporally fixed and $|E(S)|$ is maximized, where $|E(S)|$ is the set of edges whose endpoints are both in S. Their neighborhood is defined as follows. Given a vertex set $S (\subseteq V)$, a 'swap' neighborhood is defined as

$$\mathcal{N}_{swap} : S \rightarrow \{S \setminus \{x\} \cup \{y\} : x \in S, y \in V \setminus S\}.$$

Gendreau et al. [16] maximized the objective function $|C| + |A(C)|$ which is an upper bound on the size of any clique containing C, where $|A(C)|$ is the set of vertices that are adjacent to all vertices in C. Ramanujam and Sadayappan [34] proposed a heuristic using neural networks.

Another type of heuristics that finds a maximal clique of G is called the *subgraph approach* [4]. It is based on the fact that a maximum clique C of a subgraph $G' \subseteq G$ is also a clique of G. The subgraph approach first finds a subgraph $G' \subseteq G$ such that the maximum clique of G' can be found in polynomial time. Then it finds a maximum clique of G' and use it as the approximation solution. The advantage of this approach is that in finding the maximum clique $C \subseteq G'$, one has (implicitly) searched many other cliques of G' ($S_{G'} \subseteq S_G$). Because of the special structure of G', this implicit search can be done efficiently.

For more information, we refer the readers to a comprehensive survey [33] which contains more than 300 references.

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 general combinatorial optimization problem may be defined as follows.

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.

For the MSSP, the *ground set* is V. Given a set of vertices $S \subseteq V$, we denote by $E(S)$ the set of edges whose endpoints are both in S. Then the set of feasible solutions $X \subseteq 2^V$ is defined by

$$X = \{S \subseteq V : |E(S)| = 0\}. \tag{1}$$

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

$$c(S) = -|S|. \tag{2}$$

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.

For the MSSP, we introduce the set of *pseudofeasible* solutions $\tilde{X}$ which are the set of infeasible solutions that are 'close' to the feasible solutions. We define the details of the neighborhood for the MSSP in Section 4.2.

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

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

Using this mapping, a prototype of local search algorithm is described on Figure 1,

A good survey of local search procedures can be found in [31, § 18].

Although many variants of local search have been proposed, we adopt tabu search (or steepest ascent mildest descent method) introduced by Glover [18, 19] and independently by Hansen [21, 22] , 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, 8, 36] and the genetic algorithm [20, 29].

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

Figure 1: Local Search.

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 on Figure 2,

```
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     $x_{t+1} := best(x_t)$
7     $TL := TL \cup \{x_t\} \setminus \{x_{t-tabulength}\}$
8     $t := t + 1$
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; Glover recommended the following approximation. An *attribute* is the ‘coding’ or ‘finger-print’ 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 [18, 19].

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 on Figure 3,

```
procedure life span method
 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 do
 5     $x' := best(x)$
 6     $LS(\beta) := tabulength$ for all $\beta \in x \triangle x'$
 7     $x := x'$
 8     $LS(\beta) := LS(\beta) - 1$ for all $\beta \in B$ such that $LS(\beta) > 0$
 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 [18, 19] 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 [17]; 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 [27].

4 The Life Span Method for the Maximum Stable Set Problem

To design an efficient LSM tailored to the MSSP, we must determine several features of the algorithm carefully. In this section, we describe the implementation details of our heuristic algorithm for solving the MSSP. Note that we can easily construct an algorithm for the maximum clique problem, the minimum vertex cover problem, and a weighted version of these problems based on the algorithm presented below.

Our implementation is based on the Life Span Method (LSM) described in the previous section. The LSM for MSSP has the following features:

1. The search space contains infeasible solutions.

2. The algorithm simultaneously maximizes two objective functions.

3. The algorithm has two independent neighborhoods.

4. Instead of using a queue structure 'tabu list', the algorithm uses the array that we call 'life span'.

5. Long term memory (LTM) devices are used for diversifying the search.

To develop an LSM specially designed for the MSSP, the definition of the ground B is needed at first. We adopt the following simple definition: the ground set B corresponds to the set V of vertices.

4.1 Search Space

In this section, we describe the search space of the LSM. As we have mentioned in Section 1, we expand the search space into the *pseudofeasible* solutions. If two vertex sets, S_1 and S_2, satisfy $|S_1| = |S_2|$ and $|E(S_1)| > |E(S_2)|$(recall that $|E(S)|$ represents the number of edges whose endpoints are both in S) , we say that S_2 is closer to the feasible solutions than S_1. Now the *pseudofeasible* solutions, those solutions that are 'close' to the feasible solutions. The objective of the MSSP is to increase the cardinality of the set S and, simultaneously, to minimize $|E(S)|$.

4.2 Neighborhood

The most important ingredient of the LSM is the definition of the neighborhood. We define 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

$$\mathcal{N}_{add}(S) = \{S \cup \{y\} : y \in V \setminus S\} \tag{3}$$

and

$$\mathcal{N}_{drop}(S) = \{S \setminus \{x\} : x \in S\}, \tag{4}$$

respectively. If $|E(S)| = 0$, S is a feasible stable set, we use the add operation; otherwise, we use the drop operation. Thus, we can increase the cardinality of S while keeping the cardinality of $E(S)$ to be small.

As in tabu search, our algorithm moves from a solution to the 'best' solution among its neighborhoods, i.e., we select a neighbor which minimizes $|E(S)|$.

To compute the change in the cardinality of $E(S)$ efficiently, we introduce an auxiliary array δ. For each $i \in V$, $\delta(i)$ keeps the number of vertices $j \in S$ adjacent to i, i.e., $\sum_{i \in S} \delta(i) = 2|E(S)|$. The array δ can be updated in $O(|V|)$ using the algorithm presented Section 4.7.

4.3 Attributes and Life Span

To escape local optimal and to avoid searching the same solution repeatedly, we use the notion called 'life span' which corresponds to the tabu list in tabu search. Instead of using a queue implementation as in tabu search, we use a $|V|$-dimensional array LS. In this case, the set of attributes $\mathcal{A}$ corresponds to the set of vertices V. For the add neighbor $\mathcal{N}_{add}$, the symmetric difference $x \triangle x'$ of two solutions x and $x' \in \mathcal{N}_{add}(x)$ is the added vertex y in (3). Similarly, for the drop neighbor $\mathcal{N}_{drop}$, the symmetric difference is the dropped vertex x in (4). Associated with each vertex $i \in V$, we keep the life span $LS(i)$ in which we store a positive integer of the remaining iterations that vertex i is forbidden to be used. The parameter *tabulength* decides a positive number (see the next section). In add (drop) phase, we use the parameter *tabulength*1 (*tabulength*2). We decrease $LS(i)$ by 1 for each iteration. If $LS(i) = 0$, vertex i can be added or dropped. In add and drop phases, we select the vertex i^* as follows:

$$i^* := \arg\min\{\delta(i) : i \in V \setminus S, LS(i) = 0\} \tag{5}$$

and

$$i^* := \arg\max\{\delta(i) : i \in S, LS(i) = 0\}. \tag{6}$$

4.4 Randomization of *tabulength*

As we have mentioned in the previous section, *tabulength* is an important parameter of the LSM. We adopt 'randomization' of *tabulength*. Instead of the deterministic *tabulength*, we use an uniform random number in $[1, tabulength]$.

4.5 Long Term Memory

We also use the long term memory [18] to avoid cycling. The long term memory is used to diversify the search compelling regions that are not visited before. Note that a similar idea was used in the classical local search literature [31].

When vertex i is added or dropped, we increase $LTM(i)$ by 1. Values of the long term memory never decreases while searching. Equations (5) and (6) for selecting the vertex i^* are modified to incorporate the long term memory:

$$i^* := \arg\min\{\delta(i) + \alpha_1 \times LTM(i) \ / \ 1000 : i \in S \setminus V, LS(i) = 0\},$$

$$i^* := \arg\max\{\delta(i) - \alpha_2 \times LTM(i) \ / \ 1000 : i \in S, LS(i) = 0\},$$

where α_1, α_2 are parameters.

4.6 Termination Criteria

We define the termination criteria as follows. When the predetermined number of iterations '*Stop_Count*' throughout add and drop phase is exhausted without improving objective function $-|S|$, the algorithm stops.

4.7 General Description of the Life Span Method

Now, we can describe the outline of the life span method for the MSSP in Figure 4

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

5 Experimental Analyses

In this section, we report the results of our computational experiments. All computational experiments were executed on a **Hitachi 3050** with 128MB memory and the algorithm was coded in ANSI Standard C, using GNU C compiler. Running time were measured by making the system call **times** and converting to seconds. Because it is important to optimize the various parameters, preliminary experiments were performed to select the most effective parameters for the LSM. Next, we compared the best solution and running time with previous heuristics on random graphs and benchmark instances. Finally, to investigate the average behavior, we performed 40 runs of the LSM on each problem and calculated the sample mean, standard deviation, maximum and minimum of the solutions, and running time.

5.1 The Test Beds

In this section, we introduce the test beds. These can be categorized into two classes.

```
procedure outline of Life Span Method for MSSP
 1 $S = \emptyset$
 2 $\delta(i) = 0$ for all $i \in V$
 3 $z := 0$   /* $z$ keeps $|E(S)|$ */
 4 $LS(i) := 0$ for all $i \in V$
 5 while terminate-criterion $\neq$ yes do
 6     if $z = 0$ then    /* add phase */
 7         $i^* := \arg\min\{\delta(i) + \alpha_1 \times LTM(i)/1000 : i \in S \setminus V, LS(i) = 0\}$
 8         $S := S \cup \{i^*\}$
 9         $LS(i^*) := tabulength1$
10         for all $j$ adjacent to $i^*$
11             $\delta(j) := \delta(j) + 1$
12             if $j \in S$ then $z := z + 1$
13     else   /* drop phase */
14         $i^* := \arg\max\{\delta(i) - \alpha_2 \times LTM(i)/1000 : i \in S, LS(i) = 0\}$
15         $S := S \setminus \{i^*\}$
16         $LS(i^*) := tabulength2$
17         for all $j$ adjacent to $i^*$
18             $\delta(j) := \delta(j) - 1$
19             if $j \in S$ then $z := z - 1$
20     endif
21     for all $i \in V$
22         if $LS(i) > 0$ then $LS(i) := LS(i) - 1$
```

Figure 4: The LSM for the MSSP.

5.1.1 Random Graphs

The first class of graphs is the standard random graph $G(n,p)$, defined in terms of two parameters, n and p. The parameter n specifies the number of vertices in the graph; the parameter p, $0 < p < 1$, specifies the probability that any given pair of vertices constitutes an edge. (We make the decision independently for each edge pair.) This family of graphs has been studied extensively. Given parameters n and p, let X_k be a stochastic variable denoting the number of stable sets of size k as follows:

$$X_k = \binom{n}{k} (1-p)^{k(k-1)/2}. \tag{7}$$

If $Z_{n,p}$ denotes the maximum size of a clique in a random graph, then (see [6, 28]) for the threshold function $z(n,p) = 2\log_{1/(1-p)} n - 2\log_{1/(1-p)} \log_{1/(1-p)} n + 2\log_{1/(1-p)}(e/2) + 1$ and any $\epsilon > 0$, the following holds true:

$$\lim_{n\to\infty} Prob\{\lfloor z(n,p)\rfloor - 1 - \epsilon \le Z_{n,p} \le \lfloor z(n,p)\rfloor + \epsilon\} = 1.$$

5.1.2 DIMACS Benchmark Problems

The second class of graph is the DIMACS benchmark instances available in the anonymous ftp site **dimacs.rutgers.edu**. They contain hamming, johnson, keller, c-fat graphs, etc.

5.2 Parameter Optimization

In this section, we describe the experiments to optimize parameters. We do not claim that our conclusions will be applicable to all LSM implementations, but we do believe that they are applicable to the MSSP even if graphs are larger or different in character from those we studied. As we have mentioned in Section 4, the LSM for the MSSP has five parameters affecting the search behavior and the results. Since it is impossible to investigate all possible combinations of parameter values, we studied just one or two parameters at a time. We mainly show the results of the experiments performed on a benchmark instance 'johnson12-4-5', since this problem belongs to the medium class of difficulty in all problems. Similar results were obtained for other DIMACS benchmark graphs as well as several random graphs. Despite our limited experiments, they may be useful in suggesting what questions to investigate in optimizing other LSM and tabu search implementations, and we have used them as a guide in adapting the LSM to another combinatorial optimization problems.

A general strategy for parameter optimization is as follows: First, based on preliminary experiments and also on logical considerations, it was observed that algorithm performance depends more on *tabulength* than parameters for long term memory. Among tabulengths, *tabulength*1 was found to give much greater effects than *tabulength*2, as will be confirmed in the subsequent results. Based on the above observations, *tabulength's* are first optimized. When appropriate values of *tabulengths* are sought, the long term memory was not used. More specifically, an appropriate value of *tabulength*1 is sought first with a fixed value of *tabulength*2. Parameter *tabulength*2 is then optimized by fixing *tabulength*1 to the appropriate value of *tabulength*1. 'Approximate' values of parameters for the long term memory would then be sought while fixing *tabulengths* to the identified appropriate values. After appropriate values for these parameters were identified, *tabulength*1 is reoptimized since the incorporation of the long term memory tends to reduce the proper value of *tabulength*.

The five parameters of the LSM for the MSSP

1. *tabulength*1 : the number of iterations during which movement of the vertex is forbidden in the **add phase**, i.e., we set $LS(i)$ when we add vertex i to S as follows:
   $LS(i) :=$ uniform integer random number in $[1, tabulength1]$.

2. *tabulength*2 : the number of iterations during which movement of the vertex is forbidden in the **drop phase**, i.e., we set $LS(i)$ when we drop vertex i from S as follows:
   $LS(i) :=$ uniform integer random number in $[1, tabulength2]$.

3. α_1 : the bias of long term memory (see Section 4.5 for details).
   $\delta(i) - \alpha_1 \times LTM(i)\ /\ 1000.$

4. α_2 : the bias of long term memory (see Section 4.5 for details).
   $\delta(i) + \alpha_2 \times LTM(i)\ /\ 1000.$

5. *Stop_Count* : the number of iterations without improvement (see Section 4.6 for details).

5.2.1 Optimization of *tabulength*1

First, let us examine the effect of varying the most important parameter, *tabulength*1. The parameter *tabulength*1 has a greater effect than *tabulength*2 on the performance and the time required in search, because the cardinality of $V \setminus S$ is larger than that of S in most graphs.

Figure 5 shows the relationship between *tabulength*1 and the best cost function value obtained. In these runs, the other parameters ($tabulength2, \alpha_1, \alpha_2, Stop\_Count$) were set to (3, 0, 0, 100000), i.e., in this case, we do not use the long term memory. We performed 10 runs from different initial solutions for each $tabulength1 \geq 1$.

Table 1 shows the average size of the maximum stable sets and its variance by changing *tabulength*1. Observe that there is a *proper* range of equivalently good values, and our chosen value of $tabulength1 = 35$ falls within that range. Similar results were obtained for other benchmark instances, as well as for several random graphs. For all graphs, large values of *tabulength*1 lead to comparatively poor results. As this figure hints, the algorithm also performs poorly for small values of *tabulength*1. For such *tabulength*1, the cycling occurs, and the LSM cannot escape from a local optima. Note that the average size of the maximum stable sets first increases, then decreases as *tabulength*1 increases. The same behavior occurs for the other graphs we tested.

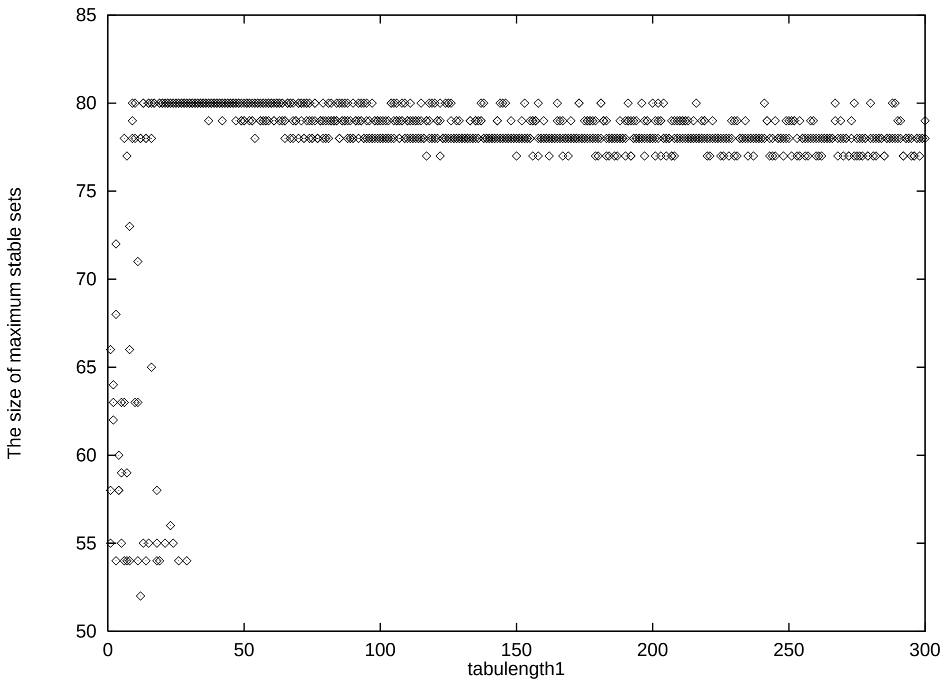

Figure 5: The relationship between *tabulength*1 and the size of the maximum stable sets on 'johnson12-4-5' while fixing the other parameters. (The *tabulength*1 increases along the X-axis. The Y-axis measures the size of the maximum stable sets.)

5.2.2 Optimization of *tabulength*2

Similarly, in this section, we optimize *tabulength*2. As we have seen, the choice of *tabulength*1 has a direct effect on the performance and we select the value of $tabulength1 = 35$. In these runs, the other parameters $(tabulength1, \alpha_1, \alpha_2, Stop\_Count)$ were set to (35, 0, 0, 100000).

Figure 6 shows the relationship between *tabulength*2 and the best size of the maximum stable sets obtained. We performed 10 runs from different initial solutions for each *tabulength*2. Because all the vertices $i \in S$ are forbidden to be used, the LSM stops when *tabulength*2 exceeds 70 on 'johnson12-4-5'.

Table 2 shows the average size of the maximum stable sets and its variance by changing *tabulength*2. The average size of the maximum stable sets increases until *tabulength*2 reaches 5 and then decreases. Based on the results, we recommend the value of $tabulength2 = 5$ that is in the middle of the proper range. But, when *tabulength*2 is between 7 and 42, the size of maximum stable sets ranges frequently from 52 to 68. For that reason, we investigate while changing *tabulength*1 and *tabulength*2 simultaneously.

Figure 7 shows the relationship between *tabulength*1, *tabulength*2 and the size of the maximum stable sets obtained. We performed 10 runs from different initial solutions for each *tabulength*1 and *tabulength*2. In this case, if both *tabulength*1 and *tabulength*2 are too small, then the LSM don't works very well without other strategies.

Table 1: Average size of the maximum stable sets (Ave) and its variance (Var) by changing *tabulength*1. Asterisk * indicates the parameter which produces best results.

Range	Ave	Var	Range	Ave	Var
1 – 10	65.0	79.2	151 – 160	78.3	0.5
11 – 20	70.7	128.3	161 – 170	78.1	0.4
21 – 30	75.8	88.3	171 – 180	78.3	0.5
31 – 40*	80.0	0.0	181 – 190	78.2	0.6
41 – 50	79.8	0.2	191 – 200	78.4	0.6
51 – 60	79.6	0.3	201 – 210	78.2	0.7
61 – 70	79.4	0.5	211 – 220	78.3	0.4
71 – 80	78.9	0.7	221 – 230	77.9	0.4
81 – 90	79.0	0.5	231 – 240	78.0	0.2
91 – 100	78.8	0.5	241 – 250	78.2	0.6
101 – 110	78.8	0.6	251 – 260	78.0	0.6
111 – 120	78.6	0.6	261 – 270	78.0	0.5
121 – 130	78.5	0.6	271 – 280	77.8	0.8
131 – 140	78.4	0.4	281 – 290	78.0	0.6
141 – 150	78.3	0.5	291 – 300	77.8	0.4

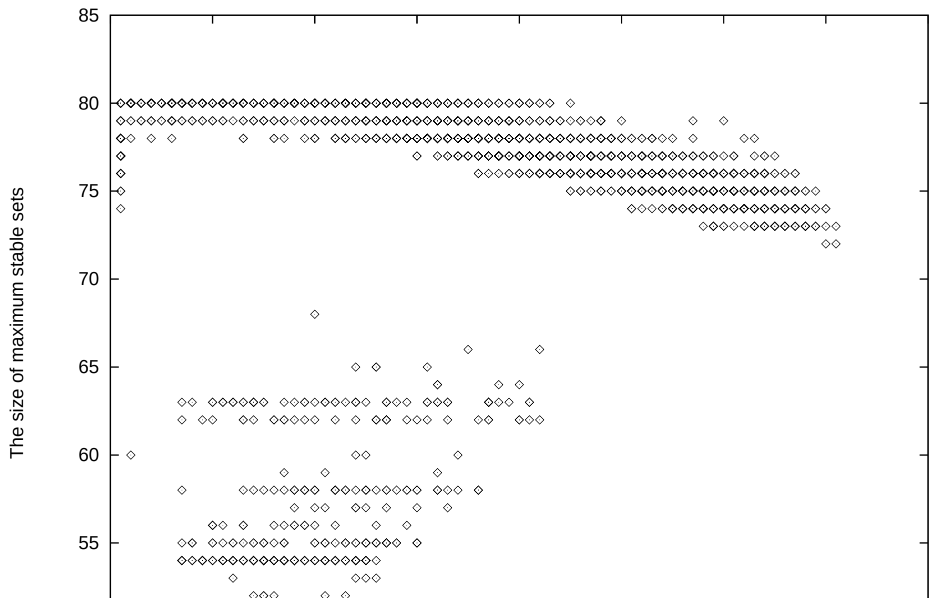

Figure 6: The relationship between *tabulength*2 and the size of the maximum stable sets on 'johnson12-4-5' while fixing the other parameters. (The *tabulength*2 increases along the X-axis. The Y-axis measures the size of the maximum stable sets.)

Table 2: Average size of the maximum stable sets (Ave) and its variance (Var) by changing *tabulength*2.

Range	Ave	Var
$1-10^*$	78.8	22.6
$11-20$	75.4	87.8
$21-30$	74.9	86.7
$31-40$	77.1	18.0
$41-50$	76.9	1.5
$51-60$	75.5	1.0
$61-70$	74.4	0.8

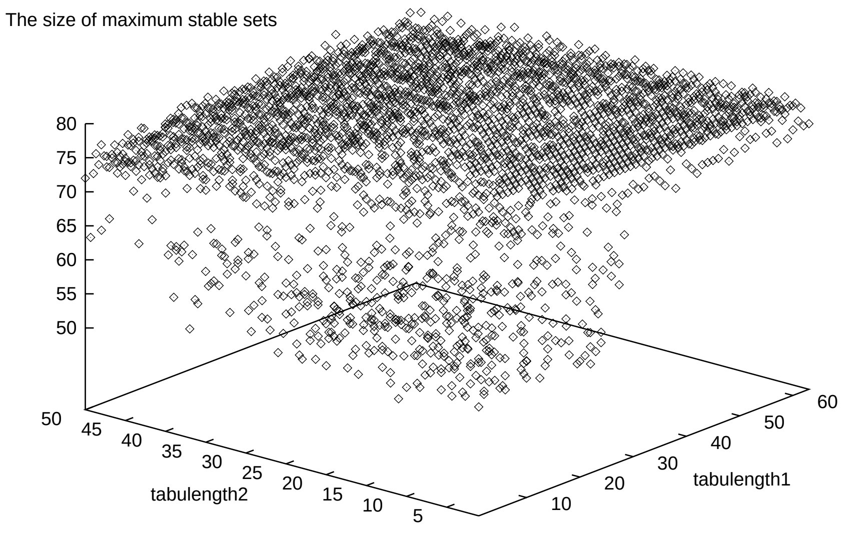

Figure 7: The relationship between *tabulength*1, *tabulength*2 and the size of the maximum stable sets on 'johnson12-4-5' while fixing the other parameters. (The *tabulength*1 and *tabulength*2 increase along the X-axis and Y-axis. The Z-axis measures the size of the maximum stable sets.)

5.2.3 Optimization of α_1

Other important parameters are α_1 and α_2, which together control the intensity of the long term memory. The other parameters $(tabulength1, tabulength2, \alpha_2, Stop\_Count)$ were set to $(35, 5, 0, 100000)$, and 10 different runs were executed from different initial solutions.

Figure 8 shows the relationship between α_1 and the best size of the maximum stable sets obtained. Table 3 shows the results of the experiments. When α_1 is between 1 and 10, the average size of the maximum stable sets is larger than other range, but as α_1 increases, the average size of the maximum stable sets decreases. If α_1 is set to 0, i.e., if we do not use the long term memory, the average size of the maximum stable sets becomes larger.

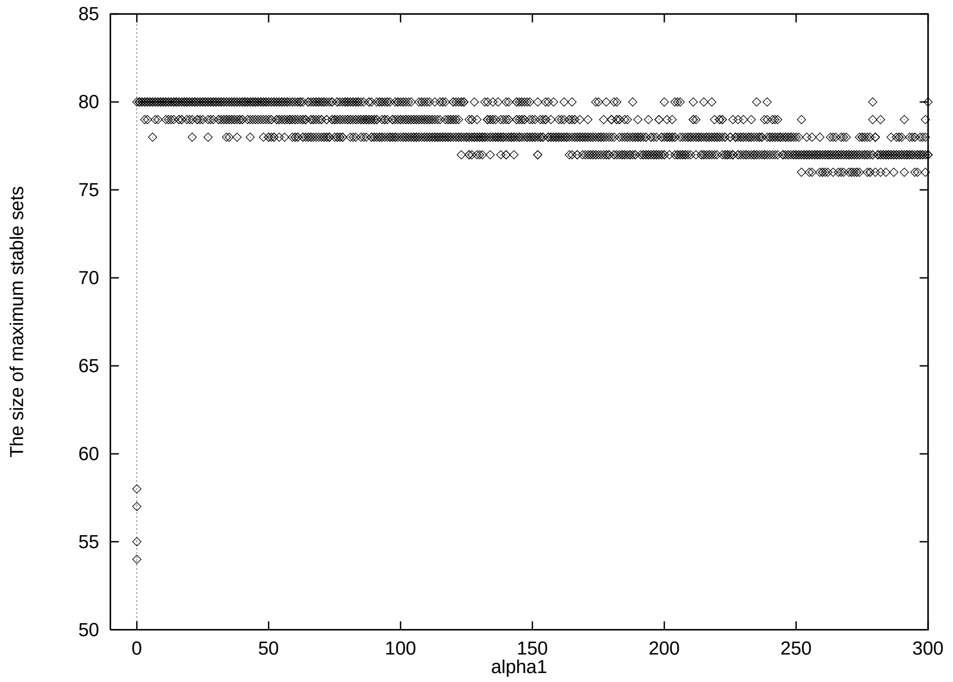

Figure 8: The relationship between α_1 and the size of the maximum stable sets on 'johnson12-4-5' while fixing the other parameters. (The α_1 increases along the X-axis. The Y-axis measures the size of the maximum stable sets.)

5.2.4 Optimization of α_2

We also performed similar experiments for optimizing the parameter α_2. The other parameters $(tabulength1, tabulength2, \alpha_1, Stop\_Count)$ were set to $(35, 5, 0, 100000)$ and we executed 10 different runs from different initial solutions for each α_2.

Figure 9 shows the relationship between α_2 and the best size of the maximum stable sets obtained. Table 4 shows the results of the experiments. As the parameter α_2 increases, the average size of the maximum stable sets decreases from left to right. We select the value of $\alpha_2 = 5$ for this graph. Similarly, we investigate while changing α_1 and α_2 simultaneously.

Figure 10 shows the relationship between α_1, α_2 and the size of the maximum stable sets obtained. We performed 10 runs from different initial solutions for each α_1 and α_2. The LSM works very well have wide proper ranges of the parameters by using both α_1 and α_2 simultaneously.

Table 3: Average size of the maximum stable sets (Ave) and its variance (Var) by changing α_1.

Range	Ave	Var	Range	Ave	Var
0	60.8	94.2	$151 - 160$	78.3	0.5
$1 - 10^*$	79.9	0.1	$161 - 170$	78.2	0.5
$11 - 20$	79.8	0.2	$171 - 180$	77.9	0.8
$21 - 30$	79.7	0.3	$181 - 190$	77.9	0.9
$31 - 40$	79.5	0.4	$191 - 200$	77.5	0.5
$41 - 50$	79.7	0.4	$201 - 210$	77.8	0.7
$51 - 60$	79.3	0.5	$211 - 220$	78.0	0.7
$61 - 70$	79.2	0.7	$221 - 230$	77.8	0.5
$71 - 80$	79.1	0.7	$231 - 240$	77.8	0.6
$81 - 90$	79.1	0.5	$241 - 250$	77.7	0.4
$91 - 100$	78.9	0.7	$251 - 260$	77.0	0.4
$101 - 110$	78.8	0.6	$261 - 270$	77.0	0.3
$111 - 120$	78.6	0.6	$271 - 280$	77.1	0.8
$121 - 130$	78.4	0.8	$281 - 290$	77.1	0.3
$131 - 140$	78.3	0.6	$291 - 300$	77.3	0.6
$141 - 150$	78.6	0.7	-	-	-

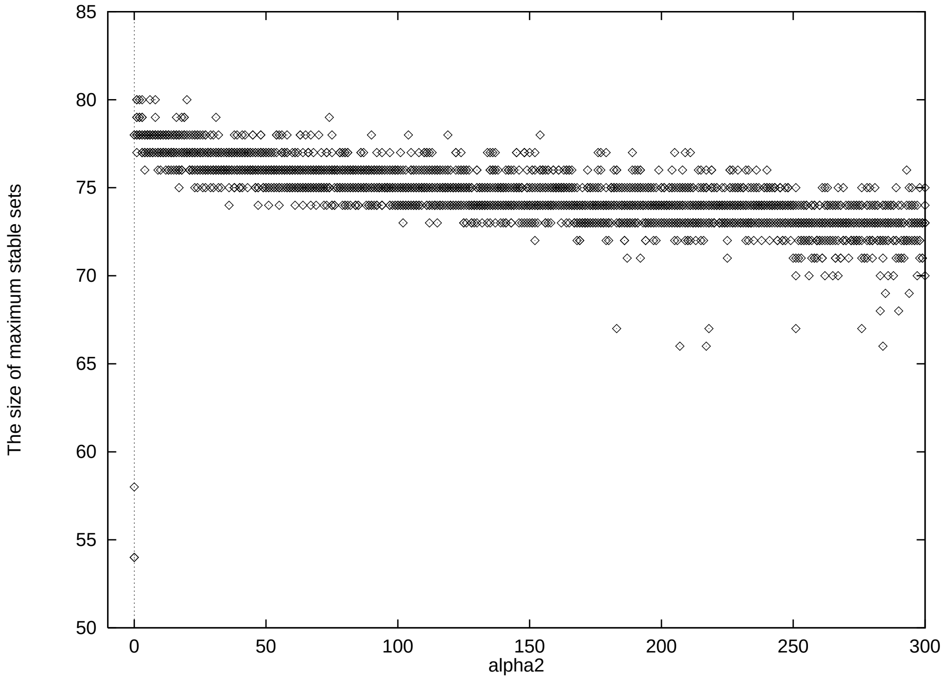

Figure 9: The relationship between α_2 and the size of the maximum stable sets on 'johnson12-4-5' while fixing the other parameters. (The α_2 increases along the X-axis. The Y-axis measures the size of the maximum stable sets.)

Table 4: Average size of the maximum stable sets (Ave) and its variance (Var) by changing α_2.

range	Ave	Var	range	Ave Cost	Var
0	64.4	125.4	$151-160$	74.1	1.1
$1-10^*$	77.9	0.8	$161-170$	74.2	1.0
$11-20$	77.3	0.8	$171-180$	73.4	1.1
$21-30$	76.6	0.7	$181-190$	72.2	1.7
$31-40$	76.3	0.7	$191-200$	73.2	0.9
$41-50$	76.3	0.7	$201-210$	73.7	1.9
$51-60$	76.0	0.8	$211-220$	73.2	2.4
$61-70$	75.7	0.9	$221-230$	73.2	0.8
$71-80$	75.6	0.9	$231-240$	73.7	0.9
$81-90$	75.4	0.7	$241-250$	72.2	0.9
$91-100$	75.2	0.6	$251-260$	72.4	1.5
$101-110$	75.0	0.9	$261-270$	72.3	1.4
$111-120$	75.0	0.9	$271-280$	72.3	1.5
$121-130$	74.7	0.9	$281-290$	72.7	2.7
$131-140$	74.6	0.9	$291-300$	72.0	1.7
$141-150$	74.7	1.1	-	-	

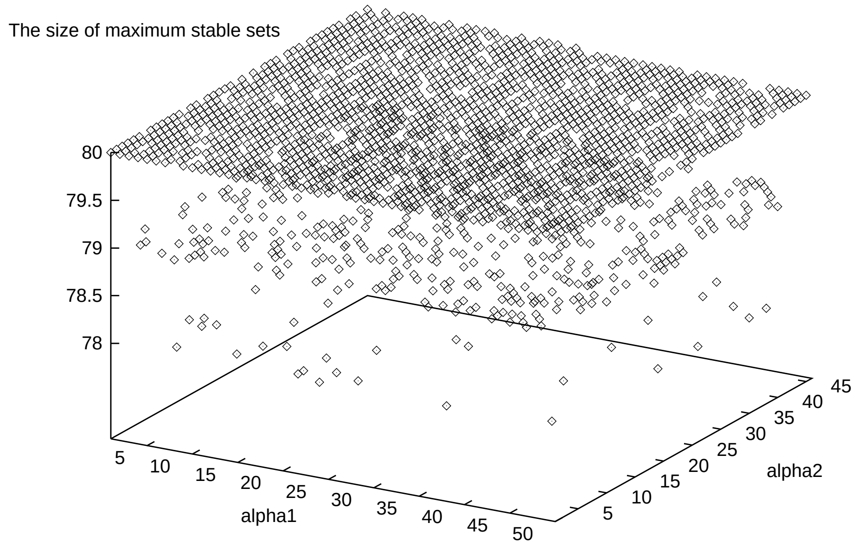

Figure 10: The relationship between α_1, α_2 and the size of the maximum stable sets on 'johnson12-4-5' while fixing the other parameters. (The α_1 and α_2 increase along the X-axis and Y-axis. The Z-axis measures the size of the maximum stable sets.)

5.2.5 Optimization of *Stop_Count*

The final parameter to be investigated is *Stop_Count*, the number of iterations which controls the termination-criterion (see Section 4.6). Note that preliminary experiments showed that the total number of iterations before termination was proportional to parameter *Stop_Count*.

Figure 11 shows the relationship between *Stop_Count* and the best size of the maximum stable sets obtained and Table 5 the results of the experiments. We performed 10 runs from different initial solutions for each *Stop_Count*. As the *Stop_Count* increases, the average size of the maximum stable sets increases.

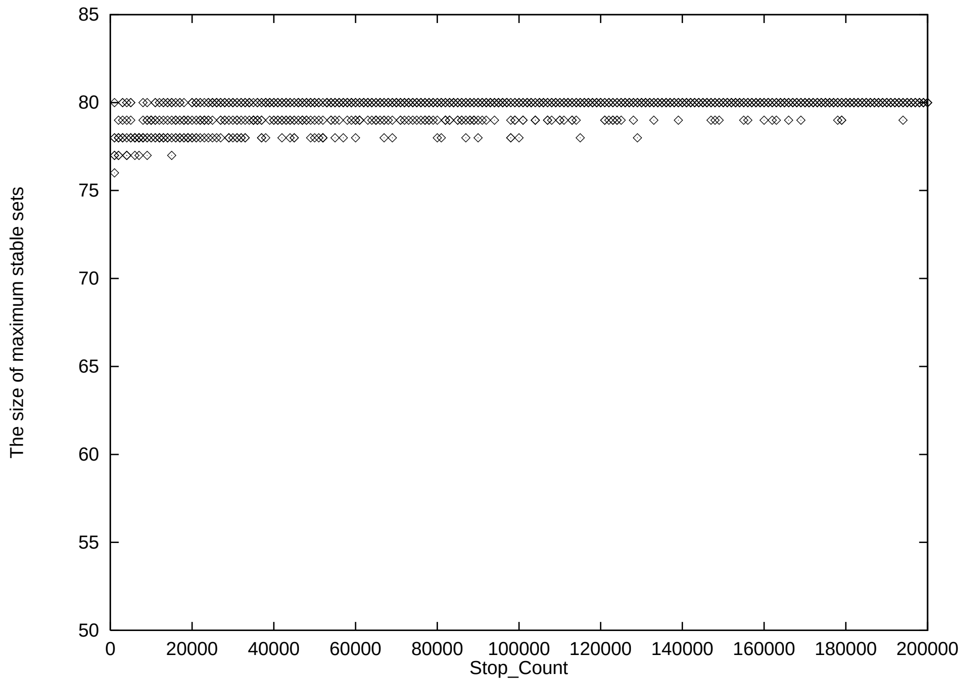

Figure 11: The relationship between *Stop_Count* and size of the maximum stable sets on 'johnson12-4-5' while fixing the other parameters. (The *Stop_Count* increases along the X-axis. The Y-axis measures the size of the maximum stable sets.)

Table 5: Average size of the maximum stable sets (Ave) and its variance (Var) by changing *Stop_Count*.

Range	Ave	Var	range	Ave	Var
$1000 - 20000$	78.5	0.9	$101000 - 120000$	79.8	0.2
$21000 - 40000$	79.4	0.6	$121000 - 140000$	79.9	0.1
$41000 - 60000$	79.5	0.5	$141000 - 160000$	79.9	0.1
$61000 - 80000$	79.7	0.2	$161000 - 180000$	79.9	0.1
$81000 - 100000$	79.7	0.2	$181000 - 200000^*$	80.0	0.0

5.2.6 Experiments Incorporating All Strategies

We perform the same experiments as in Section 5.2.1 by incorporating all the parameters optimized as above. We set $(tabulength2, \alpha_1, \alpha_2, Stop\_Count)$ to $(5, 5, 5, 200000)$ and executed the algorithm from 10 different initial solutions for each $tabulength1$. Figure 12 illustrates the relationship between $tabulength1$ and the best size of the maximum stable sets incorporating all the strategies. As can be seen from Figure 5, the LSM becomes more robust by incorporating all strategies. Table 6 shows the results of the experiments. There is a *proper* range $(11 - 20)$ of good values, and our chosen value of $tabulength = 35$ does not fall within that range, but is a good choice. Thus, so we get a proper parameter choice $(tabulength1, tabulength2, \alpha_1, \alpha_2, Stop\_Count) = (15, 5, 5, 5, 200000)$.

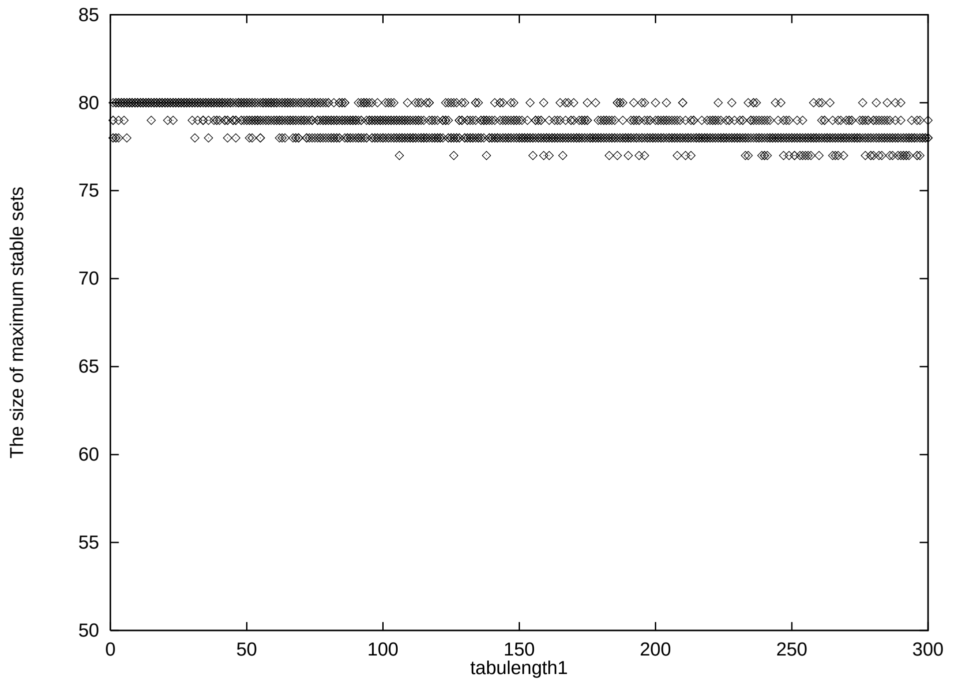

Figure 12: The relationship between $tabulength1$ and the size of the maximum stable sets incorporating all the strategies on 'johnson12-4-5' while fixing the other parameters. (The $tabulength1$ increases along the X-axis. The Y-axis measures the size of the maximum stable sets.)

5.2.7 Optimized Parameters on Benchmark Instances

Similar experiments of parameter optimization were also performed on all benchmark instances. Tables 7 and 8 show the resultant optimized parameters. We select the *proper* ranges which yield good solutions with low variances.

Table 6: Average size of the maximum stable sets (Ave) and its variance (Var) by changing *tabulength*1 incorporating all the strategies.

Range	Ave	Var	Range	Ave	Var
$3-10$	79.7	0.4	$151-160$	78.0	0.4
$11-20^*$	80.0	0.0	$161-170$	77.8	0.5
$21-30$	79.9	0.1	$171-180$	77.9	0.4
$31-40$	79.8	0.2	$181-190$	77.9	0.6
$41-50$	79.5	0.3	$191-200$	77.8	0.5
$51-60$	79.4	0.4	$201-210$	77.8	0.5
$61-70$	79.2	0.5	$211-220$	78.0	0.3
$71-80$	79.1	0.5	$221-230$	77.7	0.3
$81-90$	78.9	0.4	$231-240$	77.9	0.6
$91-100$	78.9	0.6	$241-250$	77.7	0.5
$101-110$	78.7	0.5	$251-260$	77.7	0.5
$111-120$	78.6	0.6	$261-270$	77.8	0.5
$121-130$	78.6	0.6	$271-280$	77.5	0.4
$131-140$	78.5	0.5	$281-290$	77.5	0.6
$141-150$	78.5	0.4	$291-300$	77.7	0.3

5.3 Numerical Experiments

In this section, we give the results of numerical experiments using parameters optimized using the procedure similar to the ones given.

We first test our algorithm on random graphs. We select the model which consists of graphs in which the edges are chosen independently with probability p (see [7, 30]). 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.

5.3.1 Random Graphs

In this section, we give results of the experiments on randomly generated graphs. Here, the LSM is run with the 'optimized' parameter values for random graphs obtained from a series of experiments similar to the ones given in in Section 5.2. Specifically, parameters are set as follows:

$$(tabulength1, tabulength2, \alpha_1, \alpha_2) = (10, 5, 10, 5).$$

The efficient heuristics known in the literature are tabu search algorithms due to Friden et al. [13] and Gendreau et al. [16]. Feo et al. [11] 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. [24]. We compare the performance of the LSM with their methods.

As mentioned in Section 5.1.1, random graphs have two parameters (n, p). If two graphs have the same parameters, the probabilistic estimates of the maximum size of stable sets for two graphs are identical. In these experiments, since we could not obtain the same graphs, we generated the random graphs with the same parameters. Because the computational

Table 7: The proper ranges and values of LSM parameters on benchmark instances (1).

file	$tabulength1$	$tabulength2$	α_1	α_2
c-fat200-1.clq	$11-20$	$1-10$	10	5
c-fat200-2.clq	$11-30$	$1-10$	10	5
c-fat200-5.clq	$11-40$	$1-10$	10	5
c-fat500-1.clq	$11-30$	$1-10$	10	5
c-fat500-10.clq	$11-40$	$1-10$	10	5
c-fat500-2.clq	$11-30$	$1-10$	10	5
c-fat500-5.clq	$11-50$	$1-10$	10	5
johnson16-2-4.clq	$11-100$	$1-10$	10	5
johnson32-2-4.clq	$11-100$	$1-10$	10	5
johnson8-2-4.clq	$11-100$	$1-10$	10	5
johnson8-4-4.clq	$11-100$	$1-10$	10	5
johnson12-4-5.clq	$11-20$	$11-20$	20	15
keller4.clq	$1-10$	$1-10$	10	5
keller5.clq	$1-10$	$1-10$	10	5
keller6.clq	$11-20$	$1-10$	20	10
keller7.clq	$21-30$	$11-20$	20	10
hamming11-4.clq	$1-10$	$1-10$	10	5
hamming10-2.clq	$1-10$	$1-10$	10	5
hamming10-4.clq	$1-10$	$1-10$	10	5
hamming6-2.clq	$1-10$	$1-10$	10	5
hamming6-4.clq	$11-20$	$1-10$	10	5
hamming8-2.clq	$11-20$	$1-10$	10	5
hamming8-4.clq	$11-20$	$1-10$	10	5
san1000.clq	$11-20$	$11-20$	10	5
san200_0.7_1.clq	$11-20$	$1-10$	10	5
san200_0.7_2.clq	$11-20$	$1-10$	10	5
san200_0.9_1.clq	$11-20$	$1-10$	10	5
san200_0.9_2.clq	$21-30$	$1-10$	10	5
san200_0.9_3.clq	$11-20$	$1-10$	10	5
san400_0.5_1.clq	$11-20$	$1-10$	10	5
san400_0.7_1.clq	$11-20$	$1-10$	10	5
san400_0.7_2.clq	$11-20$	$1-10$	10	5
san400_0.7_3.clq	$11-20$	$1-10$	10	5
san400_0.9_1.clq	$1-10$	$1-10$	10	5
sanr200_0.7.clq	$1-10$	$1-10$	10	5
sanr200_0.9.clq	$1-10$	$1-10$	10	5
sanr400_0.5.clq	$1-10$	$1-10$	10	5
sanr400_0.7.clq	$1-10$	$1-10$	10	5
brock200_1.clq.b	$1-10$	$1-10$	10	5
brock200_2.clq.b	$1-10$	$1-10$	10	5
brock200_3.clq.b	$1-10$	$1-10$	10	5
brock200_4.clq.b	$1-10$	$1-10$	10	5

Table 8: The proper ranges and values of LSM parameters on benchmark instances (2).

file	*tabulength*1	*tabulength*2	α_1	α_2
brock400_1.clq.b	$1-10$	$1-10$	10	5
brock400_2.clq.b	$1-10$	$1-10$	10	5
brock400_3.clq.b	$1-10$	$1-10$	10	5
brock400_4.clq.b	$1-10$	$1-10$	10	5
brock800_1.clq.b	$1-10$	$1-10$	10	5
brock800_2.clq.b	$1-10$	$1-10$	10	5
brock800_3.clq.b	$1-10$	$1-10$	10	5
brock800_4.clq.b	$1-10$	$1-10$	10	5
p_hat300-1.clq	$1-10$	$1-10$	10	5
p_hat300-2.clq	$1-10$	$1-10$	10	5
p_hat300-3.clq	$1-10$	$1-10$	10	5
p_hat500-1.clq	$1-10$	$1-10$	10	5
p_hat500-2.clq	$1-10$	$1-10$	10	5
p_hat500-3.clq	$1-10$	$1-10$	10	5
p_hat700-1.clq	$1-10$	$1-10$	10	5
p_hat700-2.clq	$1-10$	$1-10$	10	5
p_hat700-3.clq	$1-10$	$1-10$	10	5
p_hat1000-1.clq	$1-10$	$1-10$	10	5
p_hat1000-2.clq	$1-10$	$1-10$	10	5
p_hat1000-3.clq	$1-10$	$1-10$	10	5
p_hat1500-1.clq	$1-10$	$1-10$	10	5
p_hat1500-2.clq	$1-10$	$1-10$	10	5
p_hat1500-3.clq	$1-10$	$1-10$	10	5
MANN_a27.clq	$1-10$	$1-10$	10	5
MANN_a45.clq	$11-20$	$1-10$	10	5
MANN_a81.clq	$11-20$	$1-10$	20	5
MANN_a9.clq	$1-10$	$1-10$	10	5
C125.9.clq.b	$1-10$	$1-10$	10	5
C250.9.clq.b	$1-10$	$1-10$	10	5
C500.9.clq.b	$1-10$	$1-10$	10	5
C1000.9.clq.b	$1-10$	$1-10$	10	5
C2000.5.clq.b	$11-20$	$1-10$	10	5
C2000.9.clq.b	$11-20$	$1-10$	10	5
C4000.5.clq.b	$11-20$	$1-10$	10	5
DSJC500.5.clq.b	$1-10$	$1-10$	10	5
DSJ1000.5.clq.b	$1-10$	$1-10$	10	5
gen200_p0.9_44.clq.b	$1-10$	$1-10$	10	5
gen200_p0.9_55.clq.b	$1-10$	$1-10$	10	5
gen400_p0.9_55.clq.b	$1-10$	$1-10$	10	5
gen400_p0.9_65.clq.b	$1-10$	$1-10$	10	5
gen400_p0.9_75.clq.b	$1-10$	$1-10$	10	5

Table 9: Results on random graphs with $p = 0.5$.

		LSM		Friden [13]		Gendreau [16]		Feo [11]		Johnson [24]	
graph : $G(n,p)$	$\hat{\beta}$	Ave	Best	Ave	Best	Ave	Best	Ave	Best	Ave	Best
$G(100, 0.5)$	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

Table 10: Average running times in seconds for random graphs with $p = 0.5$.

		LSM	Friden [13]	Gendreau [16]	Feo [11]	Johnson [24]
4 graph : $G(n,p)$	$\hat{\beta}$	Sec.	Sec.	Sec.	Sec.	Sec.
$G(100, 0.5)$	9	0.001	1.2	20.0	-	0.001
$G(300, 0.5)$	12	0.6	62.0	30.0	-	0.05
$G(500, 0.5)$	13	1.3	50.0	50.0	-	0.11
$G(1000, 0.5)$	15	56.8	4247.0	-	241.37	0.48
$G(1500, 0.5)$	16	168.7	19009.0	-	2229.10	1.09
$G(2000, 0.5)$	17	1317.5	-	-	6609.58	2.23
$G(4000, 0.5)$	18	3346.8	-	-	-	9.73

	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

environments are different each other, running times are not directly comparable, and thus we compare the best and average sizes of the maximum stable sets. We define $\hat{\beta}$ to be the probabilistic estimates. Table 9 shows the results of experiments on random graphs with $p = 0.5$.

Table 10 show the average running times of random graphs. Friden et al.[13] 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.[16] 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.

Table 11: Results on DIMACS BENCHMARKS (LSM).

	Time(second)			Solution		
Name	Min	Avg (Std. Dev.)	Max	Min	Avg (Std. Dev.)	Max
`keller6.clq.b`	8233.92	10054.19(167.33)	15522.75	57	58.95(0.32)	59
`p_hat1500-3.clq.b`	2269.68	2292.84(15.72)	2358.50	94	94(0.00)	94
`MANN_a45.clq.b`	1360.93	1541.29(318.34)	2448.55	342	342.61(0.66)	345

Table 12: Results on DIMACS BENCHMARKS (Solution).

	Fleurent [12]			Johnson [24]		
Name	Min	Avg (Std. Dev.)	Max	Min	Avg (Std. Dev.)	Max
`keller6.clq.b`	56	56.33(0.57)	57	42	48.84(1.91)	55
`p_hat1500-3.clq.b`	93	93.66(0.57)	94	68	80.61(4.40)	91
`MANN_a45.clq.b`	342	342(0)	342	339	341.47(0.75)	344

Table 13: The average behavior on DIMACS BENCHMARKS (Time).

	Fleurent [12](minute)			Johnson [24](second)		
Name	Min	Avg (Std. Dev.)	Max	Min	Avg (Std. Dev.)	Max
`keller6.clq.b`	1447.6m	3377.8m(2740.3)	6514.3m	-	52.21(-)	-
`p_hat1500-3.clq.b`	37.6m	84.4m(44.9)	127.1m	-	4.09(-)	-
`MANN_a45.clq.b`	382.7m	430.5m(47.83)	478.3m	-	20.45(-)	-

	The computational environments
Fleurent	SUN SPARC station 10(model 50)
Johnson	SGI Challenge

5.3.2 Benchmark Instances

We also tested our algorithm on DIMACS test problems. Tables 11, 12 and 13 show the results on the DIMACS test problems. Fleurent and Ferland [12] proposed the genetic hybrid algorithms.

Though the test instances are originally for the maximum clique problem, we can obtain the stable set instances by complementing the edges. First, we performed experiments for renewing the best known solutions. Tables 14 and 15 show the results of experiments. Next, we performed experiments to investigate the average behavior. Tables 16 and 17 show the average behavior on benchmark problems. We performed 40 runs for all problems. The LSM for the MSSP is superior to other heuristics for the DIMACS test problems.

Numerical experiments show that the LSM with optimized parameters attains or renews the best known solutions. Moreover, the proposed LSM is found to be extremely stable in the sense that the best solutions could be found with almost no variability for most instances.

Table 14: Best results on DIMACS BENCHMARKS (1).

Problem	Nodes	Edges	Opt. or Best	LSM	CPU time(sec.)
c-fat200-1.clq	200	1534	12	12	0.01
c-fat200-2.clq	200	3235	24	24	0.05
c-fat200-5.clq	200	8473	58	58	0.15
c-fat500-1.clq	500	4459	14	14	0.05
c-fat500-10.clq	500	46627	126	126	0.52
c-fat500-2.clq	500	9139	26	26	0.03
c-fat500-5.clq	500	23191	64	64	0.23
johnson16-2-4.clq	120	5460	8	8	0.02
johnson32-2-4.clq	496	107880	16	16	0.03
johnson8-2-4.clq	28	210	4	4	0.01
johnson8-4-4.clq	70	1855	14	14	0.01
johnson12-4-5.clq	792	299376	≥ 80	80	190.20
keller4.clq	171	9435	11	11	0.07
keller5.clq	776	225990	27	27	17.27
keller6.clq	3361	4619898	≥ 59	59	2113.20
keller7.clq	14190	174157599	≥ 121	121	514331.72
hamming11-4.clq	2048	1859584	≥ 72	72	40.07
hamming10-2.clq	1024	518656	512	512	7.37
hamming10-4.clq	1024	434176	≥ 40	40	0.56
hamming6-2.clq	64	1824	32	32	0.03
hamming6-4.clq	64	704	4	4	0.01
hamming8-2.clq	256	31616	128	128	0.43
hamming8-4.clq	256	20864	16	16	0.01
san1000.clq	1000	250500	15	15	819.18
san200_0.7_1.clq	200	13930	30	30	5.45
san200_0.7_2.clq	200	13930	18	18	4.72
san200_0.9_1.clq	200	17910	70	70	0.28
san200_0.9_2.clq	200	17910	60	60	2.47
san200_0.9_3.clq	200	17910	44	44	1.53
san400_0.5_1.clq	400	39900	13	13	50.62
san400_0.7_1.clq	400	55860	40	40	57.15
san400_0.7_2.clq	400	55860	30	30	0.62
san400_0.7_3.clq	400	55860	22	22	6.52
san400_0.9_1.clq	400	71820	100	100	17.78
sanr200_0.7.clq	200	13868	18	18	0.12
sanr200_0.9.clq	200	17863	≥ 42	42	1.50
sanr400_0.5.clq	400	39984	13	13	8.42
sanr400_0.7.clq	400	55869	≥ 21	21	7.50
brock200_1.clq.b	200	14834	21	21	0.97
brock200_2.clq.b	200	9876	12	12	69.38
brock200_3.clq.b	200	12048	15	15	14.15
brock200_4.clq.b	200	13089	17	17	73.00

Table 15: Best results on DIMACS BENCHMARKS (2).

Problem	Nodes	Edges	Opt. or Best	LSM	CPU time(sec.)
brock400_1.clq.b	400	59723	27	27	5452.00
brock400_2.clq.b	400	59786	29	29	1453.88
brock400_3.clq.b	400	59681	31	31	670.53
brock400_4.clq.b	400	59765	33	33	296.82
brock800_1.clq.b	800	207505	23	23	1321790.63
brock800_2.clq.b	800	208166	24	24	12488.78
brock800_3.clq.b	800	207333	25	25	32217.07
brock800_4.clq.b	800	207643	26	26	20853.30
p_hat300-1.clq	300	10933	8	8	0.12
p_hat300-2.clq	300	21928	25	25	0.15
p_hat300-3.clq	300	33390	36	36	6.01
p_hat500-1.clq	500	31569	9	9	0.08
p_hat500-2.clq	500	62946	36	36	0.37
p_hat500-3.clq	500	93800	≥ 49	50	26.65
p_hat700-1.clq	700	60999	11	11	1.10
p_hat700-2.clq	700	121728	44	44	2.68
p_hat700-3.clq	700	183010	≥ 62	62	0.77
p_hat1000-1.clq	1000	122253	10	10	2.87
p_hat1000-2.clq	1000	244799	≥ 46	46	1.05
p_hat1000-3.clq	1000	371746	≥ 65	68	104.70
p_hat1500-1.clq	1500	284923	12	12	261.93
p_hat1500-2.clq	1500	568960	≥ 64	65	1.18
p_hat1500-3.clq	1500	847244	≥ 91	94	97.63
MANN_a27.clq	378	70551	126	126	0.90
MANN_a45.clq	1035	533115	345	345	127.60
MANN_a81.clq	3321	5506380	≥ 1100	1098	60.95
MANN_a9.clq	45	918	16	16	0.05
C125.9.clq.b	125	6963	34	34	1.48
C250.9.clq.b	250	27984	≥ 44	44	3.68
C500.9.clq.b	500	112332	≥ 57	57	12.85
C1000.9.clq.b	1000	450079	≥ 68	68	56.87
C2000.5.clq.b	2000	999836	≥ 16	16	39.84
C2000.9.clq.b	2000	1799532	≥ 78	78	128.43
C4000.5.clq.b	4000	4000268	≥ 18	18	356.32
DSJC500.5.clq.b	500	125248	13	13	2.45
DSJ1000.5.clq.b	1000	499652	15	15	13.43
gen200_p0.9_44.clq.b	200	17910	44	44	2.45
gen200_p0.9_55.clq.b	200	17910	55	55	2.58
gen400_p0.9_55.clq.b	400	71820	55	55	1.68
gen400_p0.9_65.clq.b	400	71820	65	65	2.46
gen400_p0.9_75.clq.b	400	71820	75	75	2.89

Table 16: The average behavior on DIMACS BENCHMARKS (1).

	Time			Solution		
Name	Min	Avg (Std. Dev.)	Max	Min	Avg (Std. Dev.)	Max
c-fat200-1.clq.b	113.77	129.25(24.88)	216.53	12	12.00(0.00)	12
c-fat200-2.clq.b	112.72	153.49(40.41)	248.35	22	23.65(0.64)	24
c-fat200-5.clq.b	109.27	119.56(16.60)	165.00	58	58.00(0.00)	58
c-fat500-1.clq.b	439.47	518.87(99.21)	1015.68	13	13.98(0.15)	14
c-fat500-10.clq.b	423.80	492.49(81.21)	900.32	126	126.00(0.00)	126
c-fat500-2.clq.b	438.82	492.36(73.19)	705.05	26	26.00(0.00)	26
c-fat500-5.clq.b	433.15	566.86(134.39)	868.82	62	63.91(0.36)	64
johnson16-2-4.clq.b	48.92	49.07(0.20)	50.08	8	8.00(0.00)	8
johnson32-2-4.clq.b	376.53	378.22(1.17)	384.87	16	16.00(0.00)	16
johnson8-2-4.clq.b	0.00	0.01(0.01)	0.03	4	4.00(0.00)	4
johnson8-4-4.clq.b	22.68	22.88(0.16)	23.60	14	14.00(0.00)	14
johnson12-4-5.clq.b	835.53	1003.73(167.33)	1681.62	80	80.00(0.00)	80
keller4.clq.b	87.32	87.54(0.32)	89.05	11	11.00(0.00)	11
keller5.clq.b	822.47	834.45(7.73)	853.45	27	27.00(0.00)	27
keller6.clq.b	8233.92	10054.19(1745.84)	15522.75	57	58.95(0.32)	59
hamming11-4.clq.b	3678.25	3757.40(43.62)	3888.05	72	72.00(0.00)	72
hamming10-2.clq.b	1265.28	1274.66(14.27)	1341.32	512	512.00(0.00)	512
hamming10-4.clq.b	1306.55	1331.58(23.71)	1406.28	40	40.00(0.00)	40
hamming6-2.clq.b	20.12	20.20(0.11)	20.77	32	32.00(0.00)	32
hamming6-4.clq.b	0.00	0.01(0.01)	0.03	4	4.00(0.00)	4
hamming8-2.clq.b	143.73	144.25(1.15)	150.58	128	128.00(0.00)	128
hamming8-4.clq.b	148.00	148.62(1.17)	154.58	16	16.00(0.00)	16
san1000.clq.b	1274.27	1722.49(362.62)	2920.58	9	10.36(1.56)	15
san200-0.7-1.clq.b	104.77	122.23(13.18)	175.23	30	30.00(0.00)	30
san200-0.7-2.clq.b	100.65	114.27(5.76)	136.80	15	17.93(0.46)	18
san200-0.9-1.clq.b	100.37	103.66(4.63)	118.53	70	70.00(0.00)	70
san200-0.9-2.clq.b	100.52	103.06(2.42)	109.20	60	60.00(0.00)	60
san200-0.9-3.clq.b	100.88	102.64(0.89)	105.03	44	44.00(0.00)	44
san400-0.5-1.clq.b	316.18	411.99(74.09)	740.48	13	13.00(0.00)	13
san400-0.7-1.clq.b	300.02	459.64(119.08)	762.45	21	26.00(7.33)	40
san400-0.7-2.clq.b	278.67	342.10(63.38)	582.27	19	29.74(1.68)	30
san400-0.7-3.clq.b	305.95	330.55(23.89)	395.95	22	22.00(0.00)	22
san400-0.9-1.clq.b	290.45	349.66(28.62)	418.37	100	100.00(0.00)	100
sanr200-0.7.clq.b	108.50	109.16(0.87)	112.00	18	18.00(0.00)	18
sanr200-0.9.clq.b	103.60	104.37(0.61)	105.95	42	42.00(0.00)	42
sanr400-0.5.clq.b	311.83	322.23(9.67)	352.23	13	13.00(0.00)	13
sanr400-0.7.clq.b	301.62	305.86(3.77)	318.70	21	21.00(0.00)	21

Table 17: The average behavior on DIMACS BENCHMARKS (2).

	Time			Solution		
Name	Min	Avg (Std. Dev.)	Max	Min	Avg (Std. Dev.)	Max
brock200-1.clq.b	106.78	112.34(4.75)	122.93	21	21.00(0.00)	21
brock200-2.clq.b	105.62	137.40(29.60)	206.05	11	11.88(0.33)	12
brock200-3.clq.b	103.08	132.77(29.93)	209.73	14	14.98(0.15)	15
brock200-4.clq.b	101.73	126.24(24.33)	200.87	16	16.66(0.47)	17
brock400-1.clq.b	300.35	321.76(22.07)	393.97	25	25.00(0.00)	27
brock400-2.clq.b	301.23	337.20(62.10)	604.48	25	25.39(1.19)	29
brock400-3.clq.b	300.62	379.91(81.27)	586.42	25	27.78(2.99)	31
brock400-4.clq.b	282.48	370.73(76.64)	594.23	25	30.27(3.79)	33
brock800-1.clq.b	878.97	1074.69(169.47)	1486.08	21	21.00(0.00)	23
brock800-2.clq.b	876.07	1087.99(190.96)	1644.98	20	20.95(0.22)	24
brock800-3.clq.b	902.73	1248.60(233.98)	1956.47	21	21.80(0.40)	25
brock800-4.clq.b	884.10	1136.56(188.47)	1746.52	20	20.98(0.15)	26
p-hat300-1.clq.b	203.88	205.13(1.18)	210.32	8	8.00(0.00)	8
p-hat300-2.clq.b	187.92	188.85(0.99)	192.65	25	25.00(0.00)	25
p-hat300-3.clq.b	186.30	187.37(1.15)	191.15	36	36.00(0.00)	36
p-hat500-1.clq.b	434.33	437.68(2.70)	450.17	9	9.00(0.00)	9
p-hat500-2.clq.b	397.82	401.02(2.24)	408.07	36	36.00(0.00)	36
p-hat500-3.clq.b	393.23	396.72(3.33)	410.13	50	50.00(0.00)	50
p-hat700-1.clq.b	729.72	737.33(7.42)	757.27	11	11.00(0.00)	11
p-hat700-2.clq.b	660.47	665.52(4.61)	683.20	44	44.00(0.00)	44
p-hat700-3.clq.b	650.87	655.35(4.26)	669.58	62	62.00(0.00)	62
p-hat1000-1.clq.b	1363.87	1381.12(9.66)	1412.65	10	10.00(0.00)	10
p-hat1000-2.clq.b	1254.48	1269.52(11.12)	1308.82	46	46.00(0.00)	46
p-hat1000-3.clq.b	1235.42	1247.35(7.99)	1272.67	68	68.00(0.00)	68
p-hat1500-1.clq.b	2552.52	3118.76(523.78)	4646.03	11	11.85(0.35)	12
p-hat1500-2.clq.b	2306.40	2326.36(19.36)	2413.38	65	65.00(0.00)	65
p-hat1500-3.clq.b	2269.68	2292.84(15.72)	2358.50	94	94.00(0.00)	94
MANN-a27.clq.b	267.02	292.12(36.99)	404.22	126	126.00(0.00)	126
MANN-a45.clq.b	1360.93	1541.29(318.34)	2448.55	342	342.61(0.66)	345
MANN-a81.clq.b	7896.23	8658.61(850.55)	11323.15	1096	1097.08(0.50)	1098
MANN-a9.clq.b	12.88	12.92(0.04)	13.05	16	16.00(0.00)	16
C125.9.clq.b	52.27	52.66(0.26)	54.02	34	34.00(0.00)	34
C250.9.clq.b	145.27	146.20(0.85)	148.85	44	44.00(0.00)	44
C500.9.clq.b	413.77	534.32(107.12)	793.52	56	56.88(0.33)	57
C1000.9.clq.b	1310.42	1821.55(448.81)	2900.17	66	67.22(0.61)	68
C2000.5.clq.b	3875.17	4317.47(505.22)	5999.18	16	16.00(0.00)	16
C2000.9.clq.b	3689.77	5300.32(1387.07)	8638.13	75	75.68(0.61)	78
C4000.5.clq.b	11396.42	13346.77(2753.41)	23336.52	17	17.25(0.43)	18
DSJC500.5.clq.b	434.50	438.55(3.58)	450.17	13	13.00(0.00)	13
DSJC1000.5.clq.b	1368.27	1530.76(159.08)	2085.47	15	15.00(0.00)	15
gen200-p0.9-44.clq.b	102.87	103.46(0.47)	104.98	44	44.00(0.00)	44
gen200-p0.9-55.clq.b	97.55	99.40(1.26)	104.22	55	55.00(0.00)	55
gen400-p0.9-55.clq.b	288.32	301.58(12.91)	343.25	55	55.00(0.00)	55
gen400-p0.9-65.clq.b	274.45	278.44(2.00)	284.90	65	65.00(0.00)	65
gen400-p0.9-75.clq.b	275.75	279.25(1.74)	284.85	75	75.00(0.00)	75

6 Conclusions

We presented a simple and efficient heuristic algorithm for the maximum stable set problem. The presented algorithm is based on a variant of tabu search that we call the Life Span Method (LSM).

As can be seen from the extensive experiments described in Section 5, the LSM seems to be one of the best approaches to the maximum stable set problem. On random graphs, the LSM dominates some previous heuristics such as Friden's tabu search [13], Gendreau's tabu search [16] and Feo's GRASP [11]. Numerical experiments on benchmark instances showed that the proposed algorithm is not only fast but also robust enough; it always generates the solutions whose values are equal to the probabilistic estimates or the best known values.

Furthermore, the extensive experimental analysis gives us an insight for a good choice of parameters.

The Long Term Memory techniques reduce the proper range of *tabulength* from (30,40) to (10,20) on 'johnson12-4-5' and similar phenomena were observed such as graph coloring problem, quadratic assignment problem, graph partitioning problem and so on. For all test problems, there are wide proper ranges of parameters. This indicates that the proposed algorithm is robust and we can easily find good parameters via test runs for new data sets. The recommendation values of parameter *tabulength* are in the range $[10, 20]$ for many test problems; but the other settings may not deteriorate the performance of the LSM. Similarly, the parameter α that controls the long term memory has a broad proper range; the recommended values are in the range $[1, 10]$, but, again, the other settings would suffice.

References

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

[2] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, On the intractability of approximation problems, *Preliminary Draft, University of Rochester, NY*, 1992.

[3] S. Arora and S. Safra. Probabilistic checking of proofs; a new characterization of np. In *Proceedings 33rd IEEE Symposium on Foundations of Computer Science*, pages 2–13, IEEE Computer Society, Los Angeles, CA, 1992.

[4] E. Balas and C.S. Yu, Finding a Maximum Clique in an Arbitrary Graph, *SIAM J. Computing*, 14, No. 4: 1054-1068, 1986.

[5] P. Berman and G. Schnitger, On the complexity of approximating the independent set problem, In *Proceedings of the 1989 Symposium on Theoret. Aspects of Comp. Sci.*, Springer-Verlag, Lecture Notes in Computer Sciences 349: 256-268, 1989.

[6] B. Bollobás and P. Erdös, Cliques in Random Graphs, *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol 80, (1976) pp. 419–427.

[7] B. Bollobás. *Random Graphs.* Academic Press, 1985.

[8] 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.

[9] K. Corradi and S. Szabo, A Combinatorial Approach for Keller's Conjecture, *Periodica Mathematica Hungarica*, Vol. 21, No. 2: 95-100, 1990.

[10] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, Approximating the maximum clique is almost NP-complete, *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, 2-12, 1991.

[11] 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.

[12] C. Fleurent and J.A. Ferland, Object-Oriented implementation of heuristic search methods for graph coloring, maximum clique, ans satisfiability, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.*, 1994.

[13] 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.

[14] 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.

[15] M. Garey and D. Johnson, *Computers and Intractability, A guide to the Theory of NP-Completeness*, FREEMAN, San Francisco, 1979

[16] 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.

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

[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, 1989.

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

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

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

[23] 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.

[24] 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.

[25] N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. An interior point approach to the maximum independent set problem in dense random graphs. In *Proceedings of the XV Latin American Conference on Informatics I*, pages 241–260, 1989.

[26] R. Kopf and G. Ruhe. A computational study of the weighted independent set problem for general graphs. *Foundations of Control Engineering*, 12:167–180, 1987.

[27] 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.

[28] D. Matula, The Largest Clique Size in a Random Graph, Southern Methodist University, Tech. Report, CS 7608 (April 1976).

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

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

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

[32] C. Papadimitriou and M. Yannakakis, Optimization, approximation and complexity classes, *Proc. of the Twentieth Annual ACM STOC*, 229-234, 1988.

[33] P. M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, (to appear), 1993.

[34] J. Ramanujam and P. Sadayappanv. Optimization by neural networks. *IEEE International Conf. on Nerual Networks*, II: 325–332, July 24-27 1988.

[35] S.K. Stein, Algebraic tiling, *Amer. Math. Monthly*, 81: 445-462, 1974.

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