Iterated k-Opt Local Search for the Maximum Clique Problem

Kengo Katayama, Masashi Sadamatsu, and Hiroyuki Narihisa

Information and Computer Engineering,
Okayama University of Science
1 - 1 Ridai-cho, Okayama, 700-0005 Japan
{katayama, masashi, narihisa}@ice.ous.ac.jp

Abstract. This paper presents a simple iterated local search metaheuristic incorporating a k-opt local search (KLS), called *Iterated KLS* (*IKLS* for short), for solving the maximum clique problem (MCP). IKLS consists of three components: LocalSearch at which KLS is used, a Kick called LEC-Kick that escapes from local optima, and Restart that occasionally diversifies the search by moving to other points in the search space. IKLS is evaluated on DIMACS benchmark graphs. The results showed that IKLS is an effective algorithm for the MCP through comparisons with multi-start KLS and state-of-the-art metaheuristics.

1 Introduction

Let $G = (V, E)$ be an arbitrary undirected graph where V is the set of n vertices and $E \subseteq V \times V$ is the set of edges in G. For a subset $S \subseteq V$, let $G(S) = (S, E \cap S \times S)$ be *the subgraph* induced by S. A graph $G = (V, E)$ is *complete* if all its vertices are pairwise adjacent, i.e., $\forall\ i,\ j \in V$ with $i \neq j$, $(i, j) \in E$. A *clique* C is a subset of V such that the induced graph $G(C)$ is complete. The objective of the maximum clique problem (MCP) is to find a clique of maximum cardinality in G. See the recent survey [3] on the MCP, which also contains an extensive bibliography.

The MCP is known to be $\mathcal{NP}$-hard [4] for arbitrary graphs, and strong negative results have been shown. Håstad [6] showed that if $\mathcal{NP} \neq \mathcal{ZPP}$ then no polynomial time algorithm can approximate the maximum clique within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$, and this margin was tightened by Khot [13] to $n/2^{(\log n)^{(1-\epsilon)}}$.

Several exact methods such as branch-and-bound algorithms [23,20] have been proposed to solve the MCP exactly. However, their effectiveness and applicability are limited to relatively small or sparse graphs. Therefore much effort has been directed towards devising efficient heuristic and metaheuristic algorithms to find near-optimal solutions to large (dense) graphs within reasonable times. A collection of them can be found in [8,3]. More recently, the following promising metaheuristic algorithms have been proposed: Reactive Local Search [2], Variable Neighborhood Search [5], Steady-State Genetic Algorithm [22], and Dynamic Local Search [21].

Most of the (meta)heuristic approaches to the MCP are based on a basic heuristical principle that prefers vertices of higher degrees to vertices of lower degrees in order to find a larger clique. Heuristics based on the principle are called *greedy*. The greedy heuristic can be regarded as a local search (or local improvement) method. The basic procedure is as follows: Given a current clique CC having a single vertex $v \in V$, one of the vertices in vertex set PA of the highest degree given by $deg_{G(PA)}$ is repeatedly added to expand CC until PA is empty, where PA denotes the vertex set of possible additions, i.e., the vertices that connected to all vertices of CC, and $deg_{G(S)}(v)$ stands for the degree of a vertex $v \in S$ in the subgraph $G(S)$, where $S \subseteq V$. This greedy method was called *1-opt local search* in [9] because at each iteration t, a single vertex $v \in PA^{(t)}$ is moved to $CC^{(t)}$ to obtain a larger clique $CC^{(t+1)} := CC^{(t)} \cup \{v\}$.

The 1-opt local search has been generalized in [9] by moving multiple vertices, instead of a single vertex, at each iteration, in order to obtain a better clique. The generalized local search is called *k-opt local search* (*KLS*). In KLS, variable, not fixed, k vertices are moved to or from a current clique simultaneously at each iteration by applying a sequence of add and drop move operations. The idea of the sequential moves in KLS is borrowed from *variable depth search* (VDS) of Lin and Kernighan [12,14]. KLS has been tested on the 37 hard DIMACS benchmark graphs [8] and compared with recent metaheuristics of reactive local search [2], genetic local search and iterated local search [16]. The results showed that KLS was capable of finding better or at least competitive cliques in reasonable times.

Since KLS can be regarded as a simple local improvement tool and there is no parameter setting by user, such as one to compulsorily stop KLS [9], it can be used, without serious modification for the algorithm, as a component in metaheuristic frameworks such as iterated local search [15], memetic algorithm [19], etc. Similar concepts can be found in several metaheuristics incorporating Lin-Kernighan heuristic (VDS based local search) for the traveling salesman problem (TSP): iterated Lin-Kernighan [7], chained Lin-Kernighan [1], and genetic iterated local search [10]. For other hard problems, several metaheuristics with VDS based local search have been proposed for the graph partitioning problem [17] and for the unconstrained binary quadratic programming problem [11,18]. These algorithms are known to be one of the best available metaheuristic algorithms to the TSP and other hard problems. It therefore seems that their high search performances mainly depend on effectiveness of the local search part of which the algorithms are composed. Judging from these contributions and dependability, the applications of metaheuristics embedded with KLS to the MCP appear to be effective and promising.

In this paper we propose an iterated local search incorporating KLS, called *Iterated KLS* (*IKLS* for short) for the MCP. IKLS consists of three components: LocalSearch KLS, Kick called *Lowest-Edges-Connectivity-based Kick* (*LEC-Kick* for short), and Restart; the details can be found in Section **2**. We evaluate the performance of IKLS on the DIMACS graphs, and show that despite the simplicity of the procedure, IKLS is effective through comparisons with multi-start KLS and the promising metaheuristics described above for the MCP.

```
    procedure IKLS
    input: graph $G = (V, E)$;
    output: best clique $C_{best}$ in $G$;
    begin
1       generate $C$; compute $PA$, $OM$, and $deg_{G(PA)}$;
2       $C$ := KLS($C$, $PA$, $OM$, $deg_{G(PA)}$); $C_{best}$ := $C$;
3       repeat
4           $C$ := LEC-Kick($C$, $PA$, $OM$, $deg_{G(PA)}$);
5           $C$ := KLS($C$, $PA$, $OM$, $deg_{G(PA)}$);
6           if $|C| > |C_{best}|$ then $C_{best}$ := $C$; endif
7           if restart=true then
8               generate $C$; compute $PA, OM$, and $deg_{G(PA)}$;
9               $C$ := KLS($C$, $PA$, $OM$, $deg_{G(PA)}$);
10              if $|C| > |C_{best}|$ then $C_{best}$ := $C$; endif
11          endif
12      until terminate=true;
13      return $C_{best}$;
    end;
```

Fig. 1. The flow of Iterated KLS

2 Iterated *k*-Opt Local Search

Iterated local search [15] can be thought of as a simple and powerful metaheuristic that repeatedly applies local search technique to solutions which are obtained by kick technique (corresponding to a mutation) that escapes from previously found local optima. In this section we show an iterated local search incorporating KLS, called *Iterated k-opt Local Search* (*IKLS* for short), for the MCP.

2.1 IKLS

Given a local optimum (clique), each iteration of IKLS consists of LocalSearch at which KLS is used and Kick that escapes from local optima obtained by KLS. As the additional strategy performed occasionally, we use Restart that diversifies the search by moving to other positions in the search space. The top level flow of IKLS is shown in Figure 1.

At first (line 1), we generate a feasible solution (clique) C that contains a single vertex selected from V at random. We then compute the associated PA, OM, and $deg_{G(PA)}$[1], where OM denotes, given a current clique CC, the vertex set of one edge missing, i.e., the vertices that are connected to $|CC| - 1$ vertices of CC, provided that $CC \subseteq OM$. KLS is applied to C, and the resulting local optimum is stored as the best clique C_{best}. The loop (lines 3–12) of IKLS that includes the three components is repeated until the stopping condition (line 12) is satisfied. In our experiments, we stop IKLS when the clique with target size that is optimal (or best-known) for each graph is found or the maximum execution number of local searches is equal to $100 \times n$, where n is the number of vertices of G.

The details of LocalSearch, Kick, and Restart components are described in the following subsections.

[1] The information of PA, OM, and $deg_{G(PA)}$ is updated whenever a single vertex is moved to or from current clique during IKLS. The updating technique and data structure used in IKLS are derived from the literature [2].

```
KLS($CC$, $PA$, $OM$, $deg_{G(PA)}$)
begin
    repeat
        $CC_{prev} := CC$, $D := CC_{prev}$, $g := 0$, $g_{max} := 0$;
        $P := \{1, \ldots, n\}$; // Note that some vertices are removed only for the first iteration; see 2.3.
        repeat
            if $|PA \cap P| > 0$ then        // Add Phase
                find a vertex $v$ with $\max_{v \in \{PA \cap P\}}$ { $deg_{G(PA \cap P)}(v)$ };
                if multiple vertices with the same maximum degree are found
                then select one vertex $v$ among them randomly;
                $CC := CC \cup \{v\}$, $g := g + 1$, $P := P \setminus \{v\}$;
                if $g > g_{max}$ then $g_{max} := g$, $CC_{best} := CC$;
            else        // Drop Phase (if $\{PA \cap P\} = \emptyset$)
                find a vertex $v \in \{CC \cap P\}$ such that the resulting $|PA \cap P|$ is maximized;
                if multiple vertices with the same size of the resulting $|PA \cap P|$ are found,
                then select one vertex $v$ among them randomly;
                $CC := CC \setminus \{v\}$, $g := g - 1$, $P := P \setminus \{v\}$;
                if $v$ is contained in $CC_{prev}$ then $D := D \setminus \{v\}$;
            endif
            update $PA$, $OM$, and $deg_{G(PA)}$;
        until $D = \emptyset$;
        if $g_{max} > 0$ then $CC := CC_{best}$ else $CC := CC_{prev}$;
    until $g_{max} \leq 0$;
    return $CC$;
end;
```

Fig. 2. The pseudo code of KLS performed in IKLS

2.2 Local Search

The work of LocalSearch process in IKLS is to find local optima in a given graph G. In the process we use KLS at lines 2, 5 and 9 in Figure 1. In the following, we briefly review KLS [9], and describe simple devices given for KLS.

KLS performs several add and drop moves for a given feasible clique CC at each iteration. The add move is to add a vertex $v \in PA$ to CC if $PA \neq \emptyset$ and the drop move is to drop a vertex $v \in CC$. A set of neighbor cliques obtained by each of the 1-opt moves is called *1-opt neighborhood*.

The 1-opt neighborhood has been based in most of the existing (meta)heuristic algorithms [8,3,16,2,21] for the MCP. Therefore, it is natural to consider lager neighborhoods than the 1-opt one. However, it is confronted with several drawbacks, such as how many vertices should be moved at each iteration, because the feasible cliques and reasonable search are usually desired in local search; see [9] on details of the drawbacks. In [9], the drawbacks were removed by introducing *variable depth search* (VDS) [12,14] that is to change the size of the neighborhood adaptively so that the algorithm can effectively traverse larger search space within reasonable time. KLS determines dynamically at each iteration the value of k, where k is the number of vertices to move. KLS efficiently explores the (*variable*) *k-opt neighborhood* defined as the set of neighbors that can be obtained by a sequence of several add and drop moves that are adaptively changed in the feasible search space. More details and the basic procedure of KLS can be found in [9].

To perform effective searches with IKLS, two simple devices are given for the original KLS of [9]. The pseudo code of KLS for which the devices are given is shown in Figure 2. Note that the devices lead to no serious modifications and the basic procedure is the same as the original KLS.

The first one is to alter the process on the vertex selection rule performed in the add and drop phases of KLS. In KLS used in IKLS, we select a vertex of

```
  LEC-Kick($CC$, $PA$, $OM$, $deg_{G(PA)}$)
  begin
1    if all $i \in CC$ are disconnected to all $j \in V \backslash CC$ then
2       select a vertex $v \in V \backslash CC$ randomly; compute $PA, OM$, and $deg_{G(PA)}$;
3       $CC := \emptyset$; $CC := CC \cup \{v\}$; return new clique $CC$;
4    endif
5    find a vertex $v \in V \backslash CC$ with the lowest edge number to vertices of $CC$.
6    if multiple vertices with the same lowest edge number are found
     then select one vertex $v$ among them randomly;
7    drop vertices from $CC$ that are not connected to $v$;
     // the dropped vertices are removed from $P$ in Fig. 2 (line 3) only for 1st iteration of the next KLS.
8    update $PA, OM$, and $deg_{G(PA)}$;
9    return new clique $CC$;
  end;
```

Fig. 3. The pseudo code of Kick process in IKLS

the highest degree in subgraph $G(PA \cap P)$ at line 6 instead of $G(PA)$ in the add phase, and select a vertex from the current clique in the drop phase such that resulting $|PA \cap P|$ instead of $|PA|$ is maximized at line 11. This simple device in KLS contributes to slightly improve average size of cliques obtained by it without increasing the computation time in many cases.

The second device (line 3) is related to the following Kick process.

2.3 Kick

The role of Kick is to escape from local optima found by LocalSearch process by moving to other points where are not so far from the local optima in the search space. This moving is made by perturbing the current local optimum slightly so that a different local optimum can be found by the forthcoming local search. Therefore, the general desire in designing Kick for a specific problem is to reduce the probability to fall back into a previously found local optimum without moving to a search point where is far away from the current one.

In the traveling salesman problem (TSP), a well-known kick method called *double-bridge move* [1,7,15] has been used for iterated local search and other relevant metaheuristic approaches in which a local search such as Lin-Kernighan heuristic [14] is incorporated. It is known to be very useful in that it avoids to fall back into a local optimum just found by the previous local search.

Since the double-bridge kick cannot be applied to the MCP, we newly design a simple kick method for IKLS. One of the simplest methods is, given a current clique CC, to drop m ($1 \leq m < |CC|$) vertices from CC. However, it is difficult to determine the number m that corresponds to *perturbation strength* [15] because a suitable and optimal m may depend on graph, clique given, etc. Therefore, the perturbation strength should be determined adaptively in IKLS. Our newly designed method, called *Lowest-Edges-Connectivity-based Kick* (*LEC-Kick* for short) is quite simple, and no parameter setting value such as m is required.

In IKLS, the Kick process (LEC-Kick) is performed at line 4 in Figure 1, and the detail is shown in Figure 3. Lines 1–4 show the exceptional processing performed only when all vertices of CC have no edge to all vertices of $V \backslash CC$ in G. In this case, we select a single vertex v from $V \backslash CC$ at random, and return the new (poor) clique having v.

Otherwise, the main process of LEC-Kick is performed at lines 5–9 in Figure 3. At first, we randomly select a vertex $v \in V \backslash CC$ with the *lowest edge* number to vertices of CC. After that, all vertices that are not connected to v are removed from CC to make a new clique at line 7. The new clique contains the vertex v that is not included in the initial clique given for the kick process. In this point, containing such a vertex v into a newly *kicked* clique to the forthcoming local search may contribute to reduce the probability to fall back into the clique just discovered by the previous local search.

Although the number of vertices dropped from CC is adaptively determined in the main LEC-Kick process, it may be feared that most of the vertices are dropped and therefore the resulting size of the clique kicked is too small. We will show the additional results (in Table 1) related to this concern in Section **3**.

To further reduce the probability described above, a simple device is given for the forthcoming KLS as the second device; the vertices dropped from CC at line 7 in Figure 3 are removed from the set P at line 3 in Figure 2. This device is performed only for the first iteration of the forthcoming KLS, not for all the iterations.

Let us concentrate our attention on Kick and LocalSearch processes in IKLS. Both processes are sequentially performed in every IKLS iteration to repeatedly explore the cliques that exist around local optima. It therefore seems to be similar to the "plateau search" [2,21]. We here regard the sequential search performed by the two processes as "*sawteeth search*" instead of "plateau search" because the up-and-down motion imaged from fluctuations of the sizes of cliques searched by the sequential processes is relatively larger than that imaged from the sizes of cliques found in the *original* plateau search as in [2]. Note that restricted plateau searches are self-adaptively executed in KLS itself.

2.4 Restart

The aim of Restart, the additional strategy performed occasionally in IKLS, is to diversify the main search of IKLS, i.e., the sawteeth search described at the previous subsection, by moving compulsorily to other points in the search space. Since such diversifications should be given after the sawteeth search was performed for a while, we occasionally perform Restart process if the following condition is satisfied at line 7 in Figure 1: if no new best clique CC_{best} is found for more than $|CC_{best}|$ iterations of the sawteeth search in IKLS, where $|CC_{best}|$ is the size of the best clique CC_{best} found so far in the search. In response to this requirement, a new clique is generated at line 8 by selecting a single vertex $v \in V \backslash \{CC_{best}\}$ at random. At line 9, the new clique C, the single vertex selected, is expanded by KLS, and the sawteeth search is started again.

3 Experimental Results

In order to evaluate the performance of IKLS, we performed extensive computational experiments on the 37 hard instances of DIMACS benchmark graphs with up to 4000 nodes and up to 5506380 edges, available from the Second DIMACS Implementation Challenge[2]. All experiments were performed on Hewlett-Packard xw4300 workstation with Pentium4 3.4GHz, 4GB RAM, and Fedora Core 5, using the gcc compiler 4.11 with '-O2' option. To execute the DIMACS Machine Benchmark[3], this machine required 0.45 CPU seconds for `r300.5`, 2.78 (s) for `r400.5` and 8.32 (s) for `r500.5`.

We first compare the performance of IKLS with that of multi-start KLS (MKLS for short) because it is reasonable in that both algorithms are based on the same local search KLS, and it may be useful to see the performance difference between the algorithmic frameworks. MKLS is a simple procedure, in which repeatedly KLS is applied to newly generated cliques at random, and the best overall clique is kept and output as the result. Each MKLS iteration consists of two parts: generating an initial (poor) clique having a single vertex from V at random and KLS that is applied to the initial clique. In our experiments, this simple process is repeated until the clique with target size that is known to be optimal (or best-known) for each graph is found or $100 \times n$ iterations. The stopping condition for MKLS is the same with IKLS (see **2.1**). The remaining parameter setting that should be set for IKLS has been described at **2.4**.

Table 1 shows the results of IKLS and MKLS. Each approach was run independently for 100 times on each graph, provided MKLS was carried out 10 runs instead of 100 runs only for `MANN_a81` because of a large amount of computation times. The first two columns of the table contain the instance names and their best-known clique size "BR" ('*' if optimality is proved), respectively. In the following columns for IKLS results, we show the best found clique size "Best" with the number of times in which the best found cliques could be found by the algorithm "(#B)", the average clique size "Avg" with standard deviation "(s.dev.)", the worst clique size "Worst" with the number of times in which the worst cliques were found by the algorithm "(#W)", the average running time "Time(s)"[4] with standard deviation "(s.dev.)" in seconds in case the algorithm could find the best found cliques. The following three columns show the additional information in IKLS executions: the average steps of add moves "#add", the average number of kicks applied in a single IKLS run "#kick", and the average number of vertices dropped from current cliques in the kick process "#dk". In the remaining columns, we provide for MKLS results with the corresponding meaning which can be found in the columns of the IKLS results.

We observed in Table 1 that IKLS is capable of finding the best-known clique on all graphs, while MKLS fails on `brock400_2`, `brock800_2`, `brock800_4`, and `keller6`. It was shown that the average results "Avg" of IKLS were at least equal to, or better than those of MKLS except for `C2000.9`. In particular, the results of IKLS on `MANN_a81` and `keller6`, known to be the hardest instances in the DIMACS graph set, were considerably better. Therefore the IKLS framework for the MCP seems to be better than the multi-start framework with KLS in

[2] `http://dimacs.rutgers.edu/Challenges/`

[3] `dmclique` is available from `ftp://dimacs.rutgers.edu/pub/dsj/clique`

[4] In Table 1 (and 2) the average run times less than 0.001 seconds are shown as "$< \epsilon$".

Table 1. Results of Iterated KLS and Multistart KLS on the DIMACS benchmark graphs

		IKLS (iterated k-opt local search)								MKLS (multistart k-opt local search)		
Instance	BR	Best (#B)	Avg (s.dev.)	Worst (#W)	Time(s)	(s.dev.)	#add	#kick	#dk	Best (#B)	Avg (s.dev.)	Time(s)
C125.9	34^*	34 (100)	34.00 (0)	34 (100)	$< \epsilon$	(0.001)	197	2	7	34 (100)	34.00 (0)	$< \epsilon$
C250.9	44^*	44 (100)	44.00 (0)	44 (100)	0.009	(0.009)	1761	17	10	44 (100)	44.00 (0)	0.023
C500.9	57	57 (100)	57.00 (0)	57 (100)	2.146	(1.973)	272172	2078	12	57 (100)	57.00 (0)	3.629
C1000.9	68	68 (90)	67.90 (0.300)	67 (10)	48.322	(46.010)	3943276	24027	14	68 (81)	67.81 (0.392)	130.012
C2000.9	78	78 (1)	76.11 (0.444)	75 (4)	236.405	(214.121)	11503314	59046	16	78 (1)	76.55 (0.517)	686.926
DSJC500.5	13^*	13 (100)	13.00 (0)	13 (100)	0.052	(0.045)	1406	63	9	13 (100)	13.00 (0)	0.018
DSJC1000.5	15^*	15 (100)	15.00 (0)	15 (100)	2.361	(2.245)	21217	854	11	15 (100)	15.00 (0)	0.510
C2000.5	16	16 (100)	16.00 (0)	16 (100)	7.017	(7.521)	17703	646	12	16 (100)	16.00 (0)	1.725
C4000.5	18	18 (100)	18.00 (0)	18 (100)	2285.502	(2000.747)	1545476	51537	13	18 (96)	17.96 (0.195)	3839.559
MANN_a27	126^*	126 (100)	126.00 (0)	126 (100)	0.029	(0.027)	4036	14	10	126 (100)	126.00 (0)	0.013
MANN_a45	345^*	345 (81)	344.81 (0.392)	344 (19)	752.261	(659.438)	34252779	36028	15	345 (8)	344.08 (0.271)	71.985
MANN_a81	1100	1100 (100)	1100.00 (0)	1100 (100)	512.354	(498.422)	5336497	1698	26	1100 (2/10)	1099.20 (0.400)	11877.433
brock200_2	12^*	12 (100)	12.00 (0)	12 (100)	0.081	(0.077)	7100	392	8	12 (3)	11.03 (0.170)	0.046
brock200_4	17^*	17 (100)	17.00 (0)	17 (100)	0.169	(0.171)	30669	995	9	17 (3)	16.03 (0.170)	0.030
brock400_2	29^*	29 (72)	27.88 (1.795)	25 (28)	4.006	(4.414)	569995	10750	11	25 (100)	25.00 (0)	0.169
brock400_4	33^*	33 (100)	33.00 (0)	33 (100)	1.308	(1.144)	187337	3521	11	33 (18)	26.44 (3.073)	2.154
brock800_2	24	24 (5)	21.15 (0.653)	21 (95)	3.207	(7.924)	181434	4509	12	21 (100)	21.00 (0)	0.566
brock800_4	26	26 (11)	21.55 (1.564)	21 (89)	5.861	(12.259)	326326	8185	12	21 (100)	21.00 (0)	1.413
gen200_p0.9_44	44^*	44 (100)	44.00 (0)	44 (100)	0.008	(0.006)	1967	21	9	44 (100)	44.00 (0)	0.024
gen200_p0.9_55	55^*	55 (100)	55.00 (0)	55 (100)	0.003	(0.003)	652	6	9	55 (100)	55.00 (0)	0.004
gen400_p0.9_55	55	55 (100)	55.00 (0)	55 (100)	2.506	(2.552)	370474	3022	11	55 (76)	54.52 (0.854)	13.754
gen400_p0.9_65	65	65 (100)	65.00 (0)	65 (100)	0.012	(0.010)	1716	12	12	65 (100)	65.00 (0)	0.045
gen400_p0.9_75	75	75 (100)	75.00 (0)	75 (100)	0.010	(0.007)	1190	8	11	75 (100)	75.00 (0)	0.033
hamming8-4	16^*	16 (100)	16.00 (0)	16 (100)	$< \epsilon$	(0.001)	27	0	0	16 (100)	16.00 (0)	$< \epsilon$
hamming10-4	40	40 (100)	40.00 (0)	40 (100)	0.013	(0.009)	871	8	12	40 (100)	40.00 (0)	0.122
keller4	11^*	11 (100)	11.00 (0)	11 (100)	$< \epsilon$	(0.001)	24	0	0	11 (100)	11.00 (0)	$< \epsilon$
keller5	27	27 (100)	27.00 (0)	27 (100)	0.024	(0.031)	1528	35	11	27 (100)	27.00 (0)	0.023
keller6	59	59 (100)	59.00 (0)	59 (100)	8.055	(6.983)	178369	1844	18	57 (100)	57.00 (0)	830.578
p_hat300-1	8^*	8 (100)	8.00 (0)	8 (100)	0.002	(0.002)	125	9	5	8 (100)	8.00 (0)	0.001
p_hat300-2	25^*	25 (100)	25.00 (0)	25 (100)	0.001	(0.001)	112	1	17	25 (100)	25.00 (0)	$< \epsilon$
p_hat300-3	36^*	36 (100)	36.00 (0)	36 (100)	0.004	(0.003)	720	8	15	36 (100)	36.00 (0)	0.008
p_hat700-1	11^*	11 (100)	11.00 (0)	11 (100)	0.056	(0.053)	1173	69	7	11 (100)	11.00 (0)	0.066
p_hat700-2	44^*	44 (100)	44.00 (0)	44 (100)	0.005	(0.003)	358	2	36	44 (100)	44.00 (0)	0.004
p_hat700-3	62	62 (100)	62.00 (0)	62 (100)	0.009	(0.007)	713	4	25	62 (100)	62.00 (0)	0.007
p_hat1500-1	12^*	12 (100)	12.00 (0)	12 (100)	11.110	(10.899)	73540	3752	8	12 (100)	12.00 (0)	1.387
p_hat1500-2	65	65 (100)	65.00 (0)	65 (100)	0.039	(0.034)	1547	8	41	65 (100)	65.00 (0)	0.031
p_hat1500-3	94	94 (100)	94.00 (0)	94 (100)	0.128	(0.092)	5700	24	35	94 (100)	94.00 (0)	0.295

Table 2. Results of IKLS, RLS, DLS, VNS, and HSSGA on the DIMACS benchmark graphs

		IKLS			RLS [2]			DLS [21]			VNS [5]			HSSGA [22]		
Instance	BR	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)	Best	Avg	Time(s)
C125.9	34^*	34	34.0	$< \epsilon$	34	34.0	$< \epsilon$	34	34.0	$< \epsilon$	34	34.0	0.001	34	34.0	0.017
C250.9	44^*	44	44.0	0.009	44	44.0	0.004	44	44.0	$< \epsilon$	44	44.0	0.019	44	43.8	0.097
C500.9	57	57	57.0	2.146	57	57.0	0.526	57	57.0	0.072	57	57.0	0.317	56	54.2	1.190
C1000.9	68	68	67.9	48.322	68	68.0	7.014	68	68.0	2.551	68	68.0	6.223	66	64.1	4.073
C2000.9	78	78	76.1	236.405	78	77.5	138.638	78	77.9	111.038	78	77.2	27.321	74	71.0	33.587
DSJC500.5	13^*	13	13.0	0.052	13	13.0	0.032	13	13.0	0.007	13	13.0	0.078	13	13.0	0.202
DSJC1000.5	15^*	15	15.0	2.361	15	15.0	1.086	15	15.0	0.459	15	15.0	1.278	15	14.7	2.106
C2000.5	16	16	16.0	7.017	16	16.0	1.679	16	16.0	0.556	16	16.0	1.683	16	15.4	7.855
C4000.5	18	18	18.0	2285.502	18	18.0	367.593	18	18.0	104.147	18	18.0	373.317	17	16.8	45.223
MANN_a27	126^*	126	126.0	0.029	126	126.0	0.524	126	126.0	0.027	126	126.0	0.006	126	125.5	0.904
MANN_a45	345^*	345	344.8	752.261	345	343.6	67.145	344	344.0	29.859	345	344.5	1.823	343	342.6	18.626
MANN_a81	1100	1100	1100.0	512.354	1098	1098.0	476.659	1098	1097.9	151.716	1100	1099.3	78.661	1095	1094.2	1140.894
brock200_2	12^*	12	12.0	0.081	12	12.0	1.617	12	12.0	0.013	12	11.3	0.092	12	12.0	0.082
brock200_4	17^*	17	17.0	0.169	17	17.0	3.281	17	17.0	0.026	17	16.9	0.596	17	16.7	0.325
brock400_2	29^*	29	27.8	4.006	29	26.0	7.087	29	29.0	0.274	29	27.4	5.015	29	25.1	0.670
brock400_4	33^*	33	33.0	1.308	33	32.4	18.292	33	33.0	0.038	33	33.0	3.236	33	27.0	0.787
brock800_2	24	24	21.1	3.207	21	21.0	0.797	24	24.0	9.041	21	21.0	1.033	21	20.7	3.060
brock800_4	26	26	21.5	5.861	21	21.0	1.127	26	26.0	5.103	21	21.0	3.795	21	20.1	0.867
gen200_p0.9_44	44^*	44	44.0	0.008	44	44.0	0.006	44	44.0	$< \epsilon$	44	44.0	0.079	44	43.1	0.305
gen200_p0.9_55	55^*	55	55.0	0.003	55	55.0	0.002	55	55.0	$< \epsilon$	55	55.0	0.021	55	55.0	0.082
gen400_p0.9_55	55	55	55.0	2.506	55	55.0	0.202	55	55.0	0.014	55	54.8	3.006	53	51.4	0.522
gen400_p0.9_65	65	65	65.0	0.012	65	65.0	0.008	65	65.0	$< \epsilon$	65	65.0	0.106	65	63.8	0.488
gen400_p0.9_75	75	75	75.0	0.010	75	75.0	0.008	75	75.0	$< \epsilon$	75	75.0	0.087	75	75.0	0.550
hamming8-4	16^*	16	16.0	$< \epsilon$	16	16.0	$< \epsilon$	16	16.0	$< \epsilon$	16	16.0	$< \epsilon$	16	16.0	0.002
hamming10-4	40	40	40.0	0.013	40	40.0	0.013	40	40.0	0.004	40	40.0	0.022	40	39.0	2.914
keller4	11^*	11	11.0	$< \epsilon$	11	11.0	$< \epsilon$	11	11.0	$< \epsilon$	11	11.0	$< \epsilon$	11	11.0	0.002
keller5	27	27	27.0	0.024	27	27.0	0.028	27	27.0	0.011	27	27.0	0.033	27	26.9	1.153
keller6	59	59	59.0	8.055	59	59.0	31.96	59	59.0	97.969	59	58.2	21.509	57	54.2	89.820
p_hat300-1	8^*	8	8.0	0.002	8	8.0	0.003	8	8.0	$< \epsilon$	8	8.0	0.001	8	8.0	0.005
p_hat300-2	25^*	25	25.0	0.001	25	25.0	0.001	25	25.0	$< \epsilon$	25	25.0	$< \epsilon$	25	25.0	0.005
p_hat300-3	36^*	36	36.0	0.004	36	36.0	0.003	36	36.0	$< \epsilon$	36	36.0	0.006	36	35.9	0.051
p_hat700-1	11^*	11	11.0	0.056	11	11.0	0.031	11	11.0	0.010	11	11.0	0.045	11	11.0	0.291
p_hat700-2	44^*	44	44.0	0.005	44	44.0	0.004	44	44.0	$< \epsilon$	44	44.0	0.004	44	44.0	0.054
p_hat700-3	62	62	62.0	0.009	62	62.0	0.005	62	62.0	$< \epsilon$	62	62.0	0.005	62	61.7	0.573
p_hat1500-1	12^*	12	12.0	11.110	12	12.0	5.097	12	12.0	1.555	12	12.0	36.454	12	11.5	4.173
p_hat1500-2	65	65	65.0	0.039	65	65.0	0.026	65	65.0	0.003	65	65.0	0.024	65	64.9	0.579
p_hat1500-3	94	94	94.0	0.128	94	94.0	0.032	94	94.0	0.005	94	94.0	0.050	94	93.1	0.830

terms of solution qualities although MKLS reaches the optimal or best-known clique in shorter times on several graphs, such as DSJC1000.5, p_hat1500-1, etc.

As shown in subsection **2.3**, although the number of vertices dropped from current clique in LEC-Kick is adaptively determined with the edge density of a given graph G and the connectivity of the current clique to a vertex v selected from $V \backslash CC$ at random, it is feared that most of the vertices are dropped in LEC-Kick and the size of the resulting clique is too small. The additional results of the column "#dk" in Table 1 wipe out the fear; for example, for MANN_a81 that is a dense graph and for DSJC500.5 that is a sparse (not dense) graph with edge density of 0.5, the average numbers of vertices dropped from current cliques are 26 and 9 (that correspond to 2.36% and 69.2% in each BR of the graphs), respectively. If we suppose that the optimal clique size depends on the edge density of G and in addition, KLS obtains near-optimum cliques in many iterations of IKLS, the perturbation strength in LEC-Kick (the results of "#dk") is quite reasonable in many cases to escape from local optima.

We next compare the performance of IKLS with those of state-of-the-art promising metaheuristics with which good results have been reported: Reactive Local Search (RLS) [2], Dynamic Local Search (DLS) [21], Variable Neighborhood Search (VNS) [5], and Steady-State Genetic Algorithm (HSSGA) [22]. Table 2 summarizes the results (Best, Avg, Time(s)) of IKLS, RLS, DLS, VNS and HSSGA. To have fair comparisons, the average running times of the competitors shown in the table were all adjusted according to the results based on the DIMACS Machine Benchmark test reported in their papers.

It is considerably impressive that the cliques of size 1100 on MANN_a81, known to be one of the hardest graphs in the benchmark set, can be obtained by IKLS in *all* 100 runs in relatively short running times, while RLS and DLS only finds the cliques of size 1098 as the best result in 100 runs, and VNS obtains the average size of 1099.3 in 10 runs with the clique sizes ranging from 1098 to 1100. Therefore, the capability of IKLS to find the best-known cliques of size 1100 for MANN_a81 is superior to the recent metaheuristics. Furthermore, it can be observed that IKLS finds the best-known cliques of size 59 on keller6 in all runs in shorter running times. From these results, it should be noted that IKLS is more efficient and effective than the others for such hard graphs.

The results of IKLS for the other graphs seem to be comparable with those of RLS, DLS, and VNS except for larger and denser C graphs such as C2000.9. For larger brock graphs such as brock800_2, brock800_4, etc., DLS clearly outperforms IKLS and the others on the average clique sizes obtained.

Although in this paper the results of IKLS have been shown only for the 37 graphs chosen from the 80 DIMACS benchmark ones, additional experimental results showed that IKLS obtained the best-known cliques in all 100 runs for the remaining graphs except for larger brock graphs. In addition, we have already observed that RESTART in IKLS contributes to improve the total performance on larger MANN_a graphs in comparison to IKLS without RESTART. Finally, in IKLS we adopted the random walk acceptance criterion [15], where a new solution found by KLS is perturbed by LEC-Kick in each IKLS iteration without taking into account the quality of solutions in the acceptance decision. Additional experimental results indicated that an alternative procedure of IKLS in which C_{best} (the best clique found so far) is perturbed by LEC-Kick was inferior to IKLS shown in the paper on `C1000.9`, `C4000.5`, `MANN_a45`, `MANN_a81`, `gen400_p0.9_55`, `keller6`, and all `brock` graphs particularly.

4 Conclusion

We proposed a simple iterated local search metaheuristic, called Iterated k-opt Local Search (IKLS), for solving the maximum clique problem (MCP). Each iteration of IKLS has the components of LocalSearch at which KLS is used and a Kick called *Lowest-Edges-Connectivity-based Kick* that adaptively determines the perturbation strength. Both processes, called the *sawteeth search*, are repeated for a while, and Restart is performed occasionally in order to diversify the search by compulsorily moving to other points in the search space. After the diversification, the sawteeth search is started again. Finally the best clique (or the target sized one) found by IKLS is output as the result.

To see the performance difference between algorithmic frameworks based on the same local search, KLS, we first compared Iterated KLS (IKLS) with multi-start KLS (MKLS) on the 37 DIMACS benchmark graphs. The difference was observed on in particular hard graphs, and we showed that the IKLS framework is suitable for solving the MCP. Furthermore, the results of IKLS were compared with recent results of effective metaheuristics on the benchmark graphs. It was demonstrated that although IKLS fails on some graphs, it is capable of finding better or at least competitive cliques despite the simplicity of the procedure. In particular, it was impressive that IKLS found in all runs the cliques of size 1100 on `MANN_a81`, one of the hardest graphs in the benchmark set, and obtained the best-known cliques in all runs on `keller6` in shorter running times than the other metaheuristics. We therefore conclude that IKLS is effective and a new promising metaheuristic for the MCP.

References

1. D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, Vol. 15, No. 1, pp. 82–92, 2003.
2. R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, Vol. 29, No. 4, pp. 610–637, 2001.
3. I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization (suppl. Vol. A)*, pp. 1–74. Kluwer, 1999.
4. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.
5. P. Hansen, N. Mladenović, and D. Urošević. Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics*, Vol. 145, No. 1, pp. 117–125, 2004.

6. J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, Vol. 182, pp. 105–142, 1999.
7. D.S. Johnson and L.A. McGeoch. The traveling salesman problem: A case study. In E. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pp. 215–310. John Wiley & Sons, 1997.
8. D.S. Johnson and M.A. Trick. *Cliques, Coloring, and Satisfiability.* Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1996.
9. K. Katayama, A. Hamamoto, and H. Narihisa. An effective local search for the maximum clique problem. *Information Processing Letters*, Vol. 95, No. 5, pp. 503–511, 2005.
10. K. Katayama and H. Narihisa. Iterated local search approach using genetic transformation to the traveling salesman problem. In *Proc. of the Genetic and Evolutionary Computation Conference*, pp. 321–328, 1999.
11. K. Katayama, M. Tani, and H. Narihisa. Solving large binary quadratic programming problems by effective genetic local search algorithm. In *Proc. of the Genetic and Evolutionary Computation Conference*, pp. 643–650, 2000.
12. B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, Vol. 49, pp. 291–307, 1970.
13. S. Khot. Improved inapproximability results for maxclique, chromatic number and approximate graph coloring. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, pp. 600–609, 2001.
14. S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, Vol. 21, pp. 498–516, 1973.
15. H.R. Lourenço, O.C. Martin, and T. Stützle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, Vol. 57 of *International Series in Operations Research & Management Science*, pp. 321–353. Kluwer Academic Publishers, Norwell, MA, 2003.
16. E. Marchiori. Genetic, iterated and multistart local search for the maximum clique problem. In *Applications of Evolutionary Computing, LNCS 2279*, pp. 112–121. Springer-Verlag, 2002.
17. P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms and greedy operators for graph bi-partitioning. *Evolutionary Computation*, Vol. 8, No. 1, pp. 61–91, 2000.
18. P. Merz and K. Katayama. Memetic algorithms for the unconstrained binary quadratic programming problem. *BioSystems*, Vol. 78, No. 1–3, pp. 99–118, 2004.
19. P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, Vol. 57 of *International Series in Operations Research & Management Science*, pp. 105–144. Kluwer Academic Publishers, Norwell, MA, 2003.
20. P.R.J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, Vol. 120, No. 1–3, pp. 197–207, 2002.
21. W. Pullan and H.H. Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, Vol. 25, pp. 159–185, 2006.
22. A. Singh and A. K. Gupta. A hybrid heuristic for the maximum clique problem. *Journal of Heuristics*, Vol. 12, No. 1-2, pp. 5–22, 2006.
23. E. Tomita and T. Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In *Discrete Mathematics and Theoretical Computer Science, LNCS 2731*, pp. 278–289. Springer-Verlag, 2003.