toc.ui.ac.ir

Transactions on Combinatorics
ISSN (print): 2251-8657, ISSN (on-line): 2251-8665
Vol. 10 No. 3 (2021), pp. 149-163.

www.ui.ac.ir

A LOCAL CORE NUMBER BASED ALGORITHM FOR THE MAXIMUM CLIQUE PROBLEM

NEDA MOHAMMADI AND MEHDI KADIVAR*

Abstract. The maximum clique problem (MCP) is to determine a complete subgraph of maximum cardinality in a graph. MCP is a fundamental problem in combinatorial optimization and is noticeable for its wide range of applications. In this paper, we present two branch-and-bound exact algorithms for finding a maximum clique in an undirected graph. Many efficient exact branch and bound maximum clique algorithms use approximate coloring to compute an upper bound on the clique number but, as a new pruning strategy, we show that local core number is more efficient. Moreover, instead of neighbors set of a vertex, our search area is restricted to a subset of the set in each subproblem which speeds up clique finding process. This subset is based on the core of the vertices of a given graph. We improved the MCQ and MaxCliqueDyn algorithms with respect to the new pruning strategy and search area restriction. Experimental results demonstrate that the improved algorithms outperform the previous well-known algorithms for many instances when applied to DIMACS benchmark and random graphs.

1. Introduction

Let $G = (V, E)$ be an undirected graph, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E \subseteq V \times V$ is the set of edges. For any vertex $u \in V$, $N(u)$ is the neighbor set of u in G and $N[u] = N(u) \cup \{u\}$. The degree of a vertex u is denoted by $deg(v)$. We refer to $G[W]$ as the subgraph of G that is induced by $W \subseteq V$. A clique of G is a set of vertices, any two of which are adjacent. The maximum clique problem (MCP) is the problem of finding the clique with the largest number of vertices in G. The MCP has a wide range of practical applications in numerous fields such as, computer vision [1],

Communicated by Manouchehr Zaker.
MSC(2010): Primary: 05C69; Secondary: 05C85.
Keywords: Branch and bound, Core number, Maximum clique.
Received: 24 November 2019, Accepted: 21 January 2020.
Manuscript Type: Research Paper.
*Corresponding author.
http://dx.doi.org/10.22108/toc.2021.120153.1686

robotics [2], bioinformatics [3], computational biology [4], coding theory [5], social network analysis [6] and scheduling [7].

The MCP is an NP-hard problem [11] and also is hard to approximate [12]. So the optimal solutions cannot be reached within a reasonable time and therefore various heuristic and metaheuristic algorithms have been devised to provide approximate solutions as good as possible to large problems within an acceptable time. In [8] a survey of these algorithms for MCP is presented. In [13], a polynomial-time algorithm with an approximation guarantee of $O(n(\log\log n)^2/(\log n)^3)$ is presented.

The exact methods to solve MCP are designed based on the general branch-and-bound (B&B) technique. These methods differ from each other by their specific techniques for determining the lower and upper bounds and their branching strategies. To estimate the upper bound of the maximum clique, graph coloring techniques are used. The reason is that if a graph G can be colored with k colors, then the number of vertices in a maximum clique in G is smaller than or equal to k. However, coloring is an NP-hard problem and may be also time consuming, and therefore fast greedy coloring heuristic algorithms must be used [16, e.g. DSATUR algorithm]. An early classic B&B algorithm (denoted by BT) which employs DSATUR algorithm for finding maximum cliques can be find in [15]. Tomita et al. in [17] presented the MCQ algorithm that used the coloring of the subgraph induced by the candidate set to provide a bound on the number of vertices in a maximum clique and serve a branching strategy. The MCR [18] improved MCQ with another sorting of vertices of the input graph G. The coloring and sorting procedure of MCQ is further improved in the MaxCliqueDyn algorithm presented by Konc et al. in [19]. In MaxCliqueDyn, before coloring of the vertices, these vertices may be re-ordered non-increasingly with respect to their degrees. This process may reduce the number of used colors. The ColorSort algorithm was first described in [19] and used a pruning color threshold that vertices with color number less than that will be cut in the pruning step. The MCS [21] algorithm improved the greedy coloring procedure of MCQ algorithm, by using a recoloring strategy. Integrating this strategy in a bit string framework, lead us to an improved bit parallel exact algorithm presented by Segundo et al. in [22]. Ostergard in [14] proposed an iterative algorithm, known as Cliquer algorithm, which uses a basic idea similar to dynamic programming. Li et al. in [29] computed tighter bounds for the number of vertices in a maximum clique by reducing each colored subproblem to maximum satisfiability problem(MaxSAT). Based on this idea Segundo et al. in [28] presented an efficient way to compute related infra-chromatic upper bounds at each step of the search with no MaxSAT encoding. Segundo et al. in [20], proposed an exact bit-parallel algorithm, called BB-MaxClique, for the MCP. BB-MaxClique uses an improved approximate coloring procedure which relies on a new implicit global degree branching rule to obtain tighter upper bounds to the candidate set. Moreover, BBMCL [23] improves BB-MaxClique using a selective coloring scheme. PMC [24, 25] and FMC [26] Both use an adjacency list representation of the network and unroll the first level of the search tree to enforce early pruning. FMC employs several pruning filters for branching and uses the degree of the vertices to compute the bounds. PMC employs core numbers to estimate the bounds. Segundo et al. in [27] designed BBMCSP for large and massive sparse graphs using a novel sparse encoding for the adjacency matrix. Like PMC, in BBMCSP core

http://dx.doi.org/10.22108/toc.2021.120153.1686

numbers and coloring techniques are used to estimate the bounds. Verma et al. introduced scale reduction algorithms for large and low-density graphs which are based on clique relaxations such as k-community and k-core to find the maximum cliques and vertex colorings [34]. Li et al. in [35] presented a branch and bound algorithm, called IncMC2, which uses graph coloring and MaxSAT reasoning. Kanazawa et al. proposed an FPGA-based accelerator to solve partial MaxSAT-encoded maximum clique problems [32]. Belachew et al. introduced a continuous formulation of the MCP based on the symmetric rank-one non-negative approximation of a given matrix and showed that stationary points of the formulation are correspond to cliques of a given graph and vice versa[33]. Recently, Hungerford et al. developed a general regularization-based continuous optimization framework for the MCP [31].

In this paper, we employ smaller bounds during search which leads a faster pruning. Moreover, instead of neighbor set of $u \in V$, a subset of this set is searched to find a clique including u.

The remaining part of the paper is structured as follows. Section 2 covers notations and preliminary concepts. Section 3 describes the new algorithm and finally Section 4 shows the simulation results. Section 5 concludes the paper.

2. Basic Concepts

In this section, we define some introductory concepts and preliminaries which are used in the next sections.

The clique number $\omega(G)$ of G is the number of vertices in a maximum clique in G. Let $\Delta(G)$ and $\chi(G)$ denote the maximum degree and the chromatic number of G, respectively. Obviously, $\omega(G) \leq \chi(G) \leq \Delta(G) + 1$.

The following definition describes the concept of core and k-core as introduced by Seidman in [10].

Definition 2.1. *Let $G = (V, E)$ be a graph and $W \subseteq V$. A subgraph $H_k = G[W]$ is a k-core or a core of order k if and only if H_k is the maximal subgraph of G that all its vertices have degree at least k, $\forall u \in W$, $\deg_{H_k}(u) \geq k$. The core number of vertex u in G, denoted by $core(u)$, is the highest order of a core that contains u.*

In [9], an $O(|E|)$ time algorithm, which we call Core-algorithm, is presented to determine the core numbers of vertices of a given graph $G = (V, E)$. In this algorithm, the core number of a vertex u is $deg(u)$ if it is a vertex with minimum degree in G. Then, all edges (u, v) which $deg(u) < deg(v)$ are removed. This process is iteratively done on G until the core number of all the vertices are computed. We refer to $K_{|V|\times|V|}$ as the core-matrix of G in which $K_{uv} = 1$ for $u, v \in V$, if $v \in N(u)$ and the core number of u is computed before the core number of v by Core-algorithm. The Core-matrix algorithm computes the core number of vertices and core-matrix K (see Algorithm 3 in the Appendix).

For every $u \in V$, we define $NK(u) = \{u\} \cup \{v \mid K_{uv} = 1\}$. Clearly, for every $v \in NK(u)$ we have $core(v) \geq core(u)$.

http://dx.doi.org/10.22108/toc.2021.120153.1686

Proposition 2.2. *The core number of elements in $NK(u)$ is at least $core(u)$. Therefore, the members in $NK(u)$ have the necessary condition (i.e. the core number of each member must be at least $core(u)$) to construct a clique of size $core(u)+1$ (if it exists).*

Definition 2.3. *We refer to the core number of u in $G[N[u]]$ as the local core number of u which is denoted by $LCN(u)$. Similarly, we refer to the core number of u in $G[NK(u)]$ as the local-NK core number of u which is denoted by $LCN_{NK}(u)$.*

Note that since $NK(u) \subseteq N(u)$ we have $LCN_{NK}(u) \leq LCN(u)$.

Lemma 2.4. *Let K be core-matrix for a given graph G and $Q = \{v_1, \ldots, v_t\}$ be an arbitrary clique in G. There exist $v_i \in Q$ such that $Q \subseteq NK(v_i)$.*

Proof. Let v_i be the first vertex in Q whose core number is computed by the Core-matrix algorithm. Since v_i is the first vertex in Q considered by the algorithm, no edges of $G[Q]$ have been deleted previously, and therefore, $K_{v_iv_j} = 1$ for every $v_j \in Q$. Thus $\{v_1, \ldots, v_t\} \subseteq NK(v_i)$. □

Corollary 2.5. *Let Q_{max} be a maximum clique in G. Then, $Q_{max} \subseteq NK(u)$ in which u is the first vertex among elements in Q_{max} whose core number is computed by the Core-matrix algorithm.*

Example 1. *Let $G = (V, E)$ be the graph presented in Figure 1. In each iteration, the Core-matrix algorithm selects (if there exist more than one vertex to select, a vertex with minimum index is selected) a vertex with minimum degree in the remaining graph G. If we follow the steps of the Core-matrix algorithm then, the ordered vertices set is $\bar{V} = (v_{10}, v_{11}, v_9, v_8, v_1, v_5, v_4, v_2, v_6, v_7, v_3)$ and the K matrix is computed as follow:*

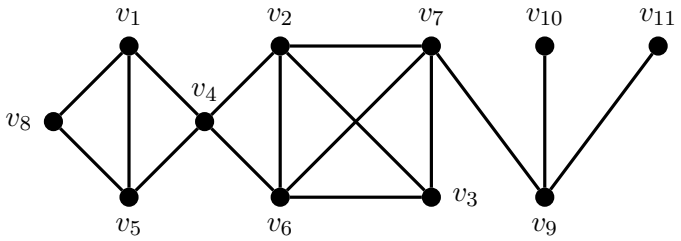

FIGURE 1. Graph G in Example 1.

http://dx.doi.org/10.22108/toc.2021.120153.1686

$$K = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \end{array} \begin{array}{c} \begin{array}{ccccccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

As the first vertex, v_{10} is selected and then $core(v_{10}) = 1$ since v_9 is the neighbor of v_{10} we have $K_{10,9} = 1$. Then, since $\deg(v_9) > \deg(v_{10})$ edge (v_{10}, v_9) is removed from G and the remaining graph is considered. In updated graph G we have $\deg(v_9) = 2$. In second iteration v_{11} is selected and we have $core(v_{11}) = 1$ and $K_{11,9} = 1$. Since $\deg(v_9) > \deg(v_{11})$ edge (v_{11}, v_9) is removed from G and in updated graph G we have $\deg(v_9) = 1$. This process is done again on G till it becomes an empty graph. The core numbers of vertices are:

$$core(v_{11}) = core(v_{10}) = core(v_9) = 1,$$

$$core(v_8) = core(v_1) = core(v_5) = core(v_4) = 2,$$

$$core(v_2) = core(v_6) = core(v_7) = core(v_3) = 3.$$

Note that for each $v_i \in V$, the i-th row of K denotes $NK(v_i) \setminus \{v_i\}$. For each clique Q in G we have $Q \subseteq NK(v_i)$ in which v_i is the first vertex among vertices in Q that its core number is computed by the Core-matrix algorithm. For example, let $Q = \{v_1, v_5, v_4\}$. We have $NK(v_1) = \{v_1, v_5, v_4\}$, $NK(v_5) = \{v_5, v_4\}$ and $NK(v_4) = \{v_4, v_2, v_6\}$. So $Q \subseteq NK(v_1)$. Moreover, as instances we can see that $NK(v_6) = \{v_6, v_3, v_7\}$, $LCN(v_6) = 3$, $LCN_{NK}(v_6) = 2$.

The existing algorithms for the maximum clique problem are branch-and-bound solvers which search in $G[N(u)]$ when vertex u is selected for branching. Corollary 2.5 shows that there exist(s) a vertex u such that $Q_{max} \subseteq NK(u)$. Therefore, instead of $N(u)$, our algorithm searches in $NK(u)$ to find Q_{max}. Since $NK(u) \subseteq N(u)$, the search area of our algorithm is smaller than the search area of the others. Moreover, instead of $\Delta(G)$, the local core numbers are used as a smaller bound for pruning.

We define $\bar{V} = \{v_1, v_2, \ldots, v_n\}$ as the ordered vertex set of a given graph G in which v_i is $i-$th vertex that its core number is computed by the Core-matrix algorithm.

Proposition 2.6. *Let $u \in \bar{V}$ and Q^u_{max} and Q^u_{NK-max} are maximum cliques in $G[N[u]]$ and $G[NK(u)]$, respectively. By the definition of core number, we can see that $|Q^u_{max}| \leq LCN(u) + 1 \leq core(u) + 1$ and*

http://dx.doi.org/10.22108/toc.2021.120153.1686

$|Q^u_{NK-max}| \leq LCN_{NK}(u)+1 \leq LCN(u)+1 \leq core(u)+1$. *Let* $LCN_{max} = \max\{LCN_{NK}(u) \mid u \in V\}$. *We have* $\omega(G) \leq LCN_{max} + 1$.

3. Maximum Clique Algorithm

In this section, we present the NK-MaxClique algorithm to find a maximum clique in a given graph $G = (V, E)$. The algorithm computes $NK(u)$ for vertex u and find a maximum clique in $G[NK(u)]$, and then the largest one of these found cliques is considred as the maximum clique of G. Algorithm 2 presents a modified version of the MCQ+CS algorithm [19] which is used in Algorithm 1 to find a maximum clique in $G[NK(u)]$.

Let Q_{max} be the set of vertices of the largest clique found until now. Initially, the NK-MaxClique algorithm puts $Q_{max} = \emptyset$. Then it finds a maximum clique in $G[NK(u)]$ by using the Modified-MCQ algorithm. By proposition 2.6 we can see that $LCN_{NK}(u) + 1$ is an upper bound for the number of vertices in a maximum clique in $G[NK(u)]$. Hence, if the inequality $|Q_{max}| \geq LCN_{NK}(u) + 1$ hols, it is implied that there is no clique larger than Q_{max} in $G[NK(u)]$). Otherwise, the algorithm finds a maximum clique in $G[NK(u)]$. In the NK-MaxClique algorithm, for every $u \in V$, the vertices of $G[NK(u)]$ are colored and sorted by the ColorSort algorithm presented in [19]. The number of colors used to color a graph is an upper bound for the clique number of the graph. It is known from [17, 30] that if the vertices of a graph be sorted non-increasingly with respect to their degree, then a tighter (smaller) bound for the clique number is obtained. Hence, in Algorithm 1, before vertex coloring, the vertices in $G[NK(u)]$ are sorted. The above descriptions lead us to the algorithm which is introduced in Algorithm 1.

Algorithm 1 *The* NK-MaxClique(G) algorithm

```
Input: undirected graph G = (V, E).
Output: A maximum clique Q_max of G.
(K, V̄) := core − matrix(G);
Q_max := ∅; for every u ∈ V̄ compute LCN(u) and LCN_NK(u).
for every u ∈ V̄ in the order do
    if |Q_max| < LCN_NK(u) + 1 then
        calculate the degrees of vertices in G[NK(u)] and sort vertices in NK(u) in a non-increasing order with respect
        to their degrees;
        C := ColorSort(G[NK(u)]);
        Modified − MCQ(NK(u), LCN, C);
    end if
end for
retun Q_max;
```

We now describe the Modified-MCQ algorithm which is presented in Algorithm 2. Let Q be the set of vertices of currently growing clique. Initially, the algorithm puts $Q = \emptyset$ and $V' = NK(u)$. Vertices are added to Q respectively until it can be verified that there are no larger clique. At each step, the vertex

http://dx.doi.org/10.22108/toc.2021.120153.1686

$v \in NK(u)$ is selected with the maximum color $C(v)$. $|Q| + C(v)$ is an upper bound for the number of vertices in a maximum clique including u and v. Moreover, proposition 2.6 indicates that $LCN(v) + 1$ is an other upper bound for the maximum clique including v. Therefore, if $|Q_{max}| < |Q| + C(v)$ and $|Q_{max}| < LCN(v) + 1$, Q is extended to $Q \cup \{v\}$. Then, V' is replaced by $V' \cap N(v)$ and this process is done recursively until Q could not be extended (for more details see [19]). The difference between the MCQ+CS algorithm and our version is that inequalities

$$LCN(v) + 1 > |Q_{max}| \tag{3.1}$$

and

$$|V' \cap N(v)| + |Q| > |Q_{max}| \tag{3.2}$$

are added to the condition of extension of Q. The formal description of the Modified-MCQ algorithm is depicted in Algorithm 2.

Algorithm 2 *The* Modified-MCQ(V', LCN, C) algorithm

```
1: Input: vertex set V' ⊆ V, Local Core number set LCN, Color set C.
2: Output: A maximum clique of G[V'].
3: while V' ≠ ∅ do
4:    Choose a vertex v with maximum color C(v) from V'.
5:    V' := V' \ {v};
6:    if LCN(v) + 1 > |Q_max| and |Q| + C(v) > |Q_max| then
7:       Q := Q ∪ {v};
8:       if V' ∩ N(v) ≠ ∅ and |V' ∩ N(v)| + |Q| > |Q_max| then
9:          C' = ColorSort(G[V' ∩ N(v)]);
10:         Modified-MCQ(V' ∩ N(v), LCN, C');
11:      else
12:         if (|Q| > |Q_max|) then
13:            Q_max := Q;
14:         end if
15:      end if
16:      Q := Q \ {v};
17:   else
18:      return;
19:   end if
20: end while
```

4. Experimental Results

In this section, we evaluate the performance of our algorithms via simulations. The MCQ+CS, MaxCliqueDyn [19], NK-MaxClique, NK-MaxCliqueDyn (see Algorithm 4 in the Appendix), and PMC [25] algorithms are considered. In the previous section, NK-MaxClique algorithm is presented in which maximum cliques are found in $NK(u)$, $u \in \bar{V}$ by Modified-MCQ algorithm (see line 9 in Algorithm 1).

http://dx.doi.org/10.22108/toc.2021.120153.1686

In NK-MaxCliqueDyn algorithm Modified-MCQ is replaced by Modified-MaxCliqueDyn algorithm (see Algorithm 5 in the Appendix). Similar to the Modified-MCQ algorithm, in the Modified-MaxCliqueDyn algorithm pruning conditions 3.1 and 3.2 are considered. In our simulations, the performance of NK-MaxClique vs MCQ+CS and the performance of NK-MaxCliqueDyn vs MaxCliqueDyn are compared. The parameter T_{limit} in MaxCliqueDyn algorithm presented in [19] is set to 0.025.

All algorithms are coded in the same programmatic style, using the same graph representation and data structures, so that there was no performance variation due to differences in implementation details. We coded the algorithms in C++ and compiled it with gcc-5.4. We have used the source code which is available for PMC (with one thread) in https://github.com/ryanrossi/pmc. We conducted the computational tests on a personal computer with a 2.67 GHz Intels Core i5 processor with 4.0 GB of memory.

Table 1 reports the output of the algorithms on DIMACS benchmark graphs and Table 2 includes the comparison of these algorithms on random graphs.

Table 1 compares the performance of the algorithms for DIMACS graphs. With respect to the CPU time and number of steps (i.e. recursive procedure calls), in most of these graphs (48 out of 57) NK-MaxClique outperforms MCQ+CS and in 40 of 57 instances NK-MaxCliqueDyn outperforms MaxCliqueDyn. The results show that NK-MaxClique is 34.51% faster than MCQ+CS in average. This value for NK-MaxCliqueDyn and MaxCliqueDyn is about 18.28%. NK-MaxCliqueDyn is atleast 50% faster than PMC. The number of steps in NK-MaxClique and in NK-MaxCliqueDyn algorithms is about 33.2% and 21.09% smaller than MCQ+CS and MaxCliqueDyn algorithms, respectively. The reason is that a stronger pruning parameters is used by the NK-MaxClique and NK-MaxCliqueDyn algorithms. In the MCQ+CS and MaxcliqueDyn algorithms $\Delta(G)+1$ colors are used for coloring the input graph. Therefore, $\Delta(G)+1$ is the upper bound for the maximum clique (pruning condition). Instead of $\Delta(G)+1$, we use $LCN_{NK}(u)+1,\ u \in V$ as the upper bound for the maximum clique. Since $LCN_{NK}(u) \leq \Delta(G)$, our pruning condition is stronger. In the c-fat family we can see $\omega(G) = LCN_{max}+1$. Therefore, comparing to the other instances, the performance of the NK-MaxClique and NK-MaxCliqueDyn algorithms is more valuable.

Table 2 reports average CPU times and number of steps of algorithms over 15 random graphs with fixed n and p, where n and p denote the number of vertices and the probability that an edge between a pair of vertices exists, respectively. The results show that NK-MaxClique and NK- MaxCliqueDyn are about 23.38% and 7.38% faster than MCQ+CS and MaxCliqueDyn, respectively. The reason is that smaller upper bounds for pruning is used in the NK-MaxClique and NK- MaxCliqueDyn algorithms.

http://dx.doi.org/10.22108/toc.2021.120153.1686

Graph	ΔG	LCN_{max}	$core_{max}$	ω	MCQ+CS[19]		NK-MaxClique		MaxCliqueDyn[19]		NK-MaxCliqueDyn		PMC [25]	
name					num.steps	CPU time	num.steps	CPU time	num.steps	CPU time	num.steps	CPU time	num.steps	CPU time
brock200_1	165	91	134	21	406,813	1.089	**317,534**	**0.857**	237,503	0.685	**231,401**	**0.664**	306,565	1.063
brock200_2	114	36	84	12	4237	0.008	**3292**	**0.007**	3606	0.008	**3295**	**0.007**	3450	0.019
brock200_3	134	57	105	15	15,850	0.042	**12,644**	**0.034**	12,717	0.034	**12,641**	**0.033**	**12,115**	0.051
brock200_4	147	69	117	17	59,081	0.138	**44,958**	**0.108**	47,762	0.113	**43,313**	**0.105**	48,745	0.155
brock400_1	320	193	277	27	286,251,929	1107.49	**223,437,337**	**874.802**	126,309,296	497.001	**118,417,294**	**474.229**	228,142,356	1095.88
brock400_2	328	194	278	29	105,226,834	471.38	**85,129,300**	**374.094**	44,365,961	210.773	**44,233,968**	**210.083**	83,743,180	453.6
brock400_3	322	193	278	31	266,993,151	944.105	**217,875,534**	**771.913**	119,316,488	427.959	**115,747,879**	**418.249**	226,831,783	1075.25
brock400_4	326	194	277	33	116,236,598	456.124	**93,922,758**	**369.462**	54,362,635	221.587	**49,873,304**	**205.926**	84,003,502	450.956
brock800_1	560	294	487	23	2,866,177,531	12512	**2,372,806,366**	**10486**	1,550,108,549	6701	**1,450,093,812**	**6406**	2,415,211,005	17904.7
brock800_2	566	294	486	24	2,382,526,517	10823	**2,087,145,977**	**9490**	**1,273,761,054**	5838	1,278,229,255	**5796**	2,123,289,010	16425.6
brock800_3	558	291	483	25	1,514,646,914	7165	**1,239,090,032**	**6029**	837,242,981	3948	**772,194,927**	**3796**	1,322,556,088	12019.9
brock800_4	565	293	485	26	1,018,936,265	5407	**807,157,439**	**4379**	544,303,947	2927	**497,578,963**	**2749**	913,078,523	5803.54
c-fat200-1	17	11	14	12	212	0.0007	**31**	**0.0002**	212	0.0006	**31**	**0.0003**	33	0.0045
c-fat200-2	34	22	32	23	217	0.0004	**44**	**0.0002**	217	0.0004	**44**	**0.0002**	69	0.0013
c-fat200-5	86	57	83	58	307	0.0028	**169**	**0.0006**	307	0.0014	**169**	**0.0006**	171	0.004
c-fat500-1	20	13	17	14	511	0.0028	**37**	**0.0008**	511	0.002	**37**	**0.0008**	39	0.007
c-fat500-2	38	25	35	26	539	0.0037	**73**	**0.0009**	539	0.0037	**73**	**0.0009**	75	0.008
c-fat500-5	95	63	92	64	617	0.0036	**187**	**0.0011**	617	0.0042	**187**	**0.0012**	189	0.045
c-fat500-10	188	125	185	126	743	0.0101	**373**	**0.0037**	743	0.0102	**373**	**0.0041**	375	0.038
hamming6-2	57	51	57	32	62	0.0004	**55**	**0.0003**	110	0.0004	**55**	**0.0003**	**32**	0.006
hamming6-4	22	7	22	4	102	**0.0002**	**100**	0.0003	104	**0.0002**	**100**	0.0003	**64**	0.003
hamming8-2	247	238	247	128	**379**	**0.0073**	584	0.0175	681	**0.0146**	**674**	0.0153	**128**	0.063
hamming8-4	163	86	163	16	21,104	0.0887	**11,781**	**0.045**	**11,293**	0.0497	11,944	**0.0449**	14,483	0.189
hamming10-2	1013	1003	1013	512	**1022**	**0.415**	2963	1.462	**9707**	**7.417**	18,292	13.316	34,220	54.681
johnson8-2-4	15	7	15	4	51	0.0001	**44**	0.0001	51	0.0001	**44**	0.0001	**29**	0.0005
johnson8-4-4	53	37	53	14	**129**	**0.0005**	229	0.001	230	**0.0007**	**219**	0.0008	**219**	0.023
johnson16-2-4	91	67	91	8	522,716	0.224	**388,831**	**0.1915**	608,065	0.303	**469,085**	**0.231**	**284,250**	0.965
keller4	124	66	102	11	17,450	0.032	**7970**	**0.0164**	8286	0.017	**7975**	**0.015**	8328	0.083
MANN_a9	41	36	40	16	94	0.0003	**92**	**0.0002**	94	0.0003	**92**	**0.0002**	**81**	0.003
MANN_a27	374	351	364	126	51,076	2.538	**47,723**	**2.478**	48,699	2.731	**45,767**	**2.62**	**42,727**	6.374
p_hat300-1	132	16	18	8	2136	0.0041	**1782**	**0.0034**	2134	0.0038	**1782**	**0.0037**	1821	0.045
p_hat300-2	229	61	66	25	7406	0.028	**5541**	**0.024**	7394	0.027	**5560**	**0.023**	5963	0.32
p_hat300-3	267	119	180	36	1,444,565	7.455	**1,021,418**	**5.292**	**630,544**	**3.512**	659,793	3.602	893,779	13.377
p_hat500-1	204	27	86	9	10,930	**0.0201**	**9496**	0.021	10,932	**0.0205**	**9498**	0.0227	**9026**	0.218
p_hat500-2	389	104	170	36	275,767	1.59	**172,752**	**1.05**	193,061	1.144	**172,703**	**1.045**	174,310	3.248
p_hat500-3	452	200	303	50	75,619,583	734.349	**49,601,214**	**494.043**	25,520,093	264.147	**25,477,611**	**252.852**	42,955,807	1163.77
p_hat700-1	286	37	117	11	32,160	0.074	**27,837**	**0.0721**	28,084	**0.0736**	**27,834**	0.0747	**26,701**	0.611
p_hat700-2	539	146	235	44	1,579,996	13.549	**1,186,704**	**10.6**	**1,092,005**	**9.539**	1,178,980	10.174	**941,696**	23.303
p_hat700-3	627	279	426	62	1,484,617,275	22106	**1,086,099,340**	**15597**	**294,372,536**	4620	294,930,306	**4473**	944,057,608	16547.2
p_hat1000-1	408	51	163	10	190,798	0.391	**171,332**	**0.368**	**170,093**	**0.364**	171,327	0.374	**167,311**	1.005
p_hat1000-2	766	202	327	46	82,151,453	785.193	**56,374,080**	**547.749**	**31,409,971**	**311.362**	33,494,037	327.79	50,126,780	1341.49
p_hat1500-1	614	78	252	12	1,213,681	3.252	**1,120,279**	**3.089**	1,136,650	3.064	**1,120,276**	**3.045**	1,127,634	8.514
san200_0.7_1	155	92	125	30	2142	0.013	**1507**	**0.006**	1199	**0.005**	**1171**	**0.005**	**519**	0.029
san200_0.7_2	164	111	122	18	**1581**	**0.005**	3205	0.009	**2222**	**0.007**	2384	0.008	**1163**	0.073
san200_0.9_1	191	134	162	70	216,631	1.059	**23,463**	**0.171**	**6746**	0.059	6978	**0.056**	**3412**	0.221
san200_0.9_2	188	143	169	60	**284,215**	**1.421**	1,081,708	4.861	84,966	0.606	**73,354**	**0.535**	301,668	4.572
san200_0.9_3	187	145	169	44	**877,124**	5.771	966,403	**5.263**	351,087	2.495	**327,680**	**2.377**	660.896	12.836
san400_0.5_1	225	153	183	13	4831	0.0232	**2476**	**0.0131**	2886	0.008	**2498**	**0.0134**	**2471**	0.095
san400_0.7_1	301	181	261	40	275,773	1.088	**71,497**	**0.415**	**40,080**	**0.266**	41,994	0.28	47,013	1.172
san400_0.7_2	304	178	259	30	127,249	1.32	**43,823**	**0.241**	**11,285**	**0.083**	19,679	0.176	30,394	0.454
san400_0.7_3	307	181	253	22	996,933	5.766	**812,063**	**3.681**	515,792	1.596	**377,992**	**1.298**	471,498	4.198
san400_0.9_1	374	293	345	100	**70,290,865**	**973.891**	411,821,994	4079	1,268,950	27.284	**462,698**	**16.151**	501,324	24.102
san1000	550	398	464	15	329,234	4.564	**132,906**	**0.347**	125,027	0.344	**108,205**	**0.325**	134,919	3.623
sanr200_0.7	161	78	124	18	167,260	0.405	**124,506**	**0.302**	**104,452**	**0.268**	105,478	0.275	122,508	0.814
sanr200_0.9	189	141	166	42	28,162,489	155.93	**15,659,491**	**86.996**	**6,129,161**	**36.93**	6,538,591	38.988	13,453,957	217.864
sanr400_0.5	233	78	177	13	294,799	0.635	**255,885**	**0.55**	255,506	0.558	**255,276**	**0.533**	**241,932**	1.385
sanr400_0.7	310	165	258	21	81,719,986	252.938	**67,788,823**	**204.239**	37,921,710	124.419	**37,917,538**	**121.586**	67,497,121	535.876

TABLE 1. CPU times and number of steps (recursive procedure calls) for DIMACS benchmark graphs. ω is the clique number, ΔG is the maximum degree. LCN_{max} and $core_{max}$ are maximum local-NK core number and maximum core number in G, respectively. The best times and minimum number of steps for each row are written in bold and underline style. Bold values in columns 6 and 7 indicate the best performance between MCQ+CS and NK-MaxClique and bold values in columns 8 and 9 indicate the priority of the MaxCliqueDyn or NK- MaxCliqueDyn algorithms. The last column indicates the performance PMC when it is run in one thread.

http://dx.doi.org/10.22108/toc.2021.120153.1686

n	p	ω	MCQ+CS		NK-MaxClique		MaxCliqueDyn		NK-MaxCliqueDyn	
			num.steps	CPU time	num.steps	CPU time	num.steps	CPU time	num.steps	CPU time
100	0.6	11-12	1049	0.0029	**800**	**0.0017**	1006	0.0021	**798**	**0.0015**
100	0.7	14-15	2780	0.007	**1928**	**0.0037**	2086	0.0045	**1925**	**0.0032**
100	0.8	19-20	5582	0.0148	**3643**	**0.0094**	3718	0.0093	**3553**	**0.0075**
100	0.9	31-32	5545	0.0195	**3674**	**0.0147**	3266	0.0105	**3176**	**0.0102**
100	0.95	41-46	**1219**	0.0069	1279	**0.0053**	1028	0.0045	**894**	**0.0041**
150	0.6	12-13	7550	0.0142	**6169**	**0.0105**	**5769**	0.0105	6161	**0.0102**
150	0.7	17	23,823	0.0506	**17,296**	**0.0359**	18,563	0.0411	**16,936**	**0.0354**
150	0.8	22-24	178,284	0.4352	**120,768**	**0.3003**	86,106	0.232	**84,900**	**0.229**
150	0.9	35,37,38	797,368	3.1861	**551,742**	**2.1814**	291,531	1.2723	**283,848**	**1.245**
150	0.95	52-54	424,686	2.5264	**158,954**	**1.048**	79,588	0.5813	**75,227**	**0.5481**
200	0.4	9	2336	0.0043	**1941**	**0.0034**	2291	0.004	**1941**	**0.0035**
200	0.5	10-11	7847	0.0145	**6281**	**0.012**	6476	0.0121	**6283**	**0.0112**
200	0.6	13-14	36,266	0.0671	**28,340**	**0.0538**	29,572	0.0559	**23,812**	**0.0536**
200	0.7	18	235,418	0.5086	**180,308**	**0.3875**	**141,997**	**0.3324**	142,158	0.3326
200	0.8	24-25	2,675,492	7.8531	**2,023,915**	**5.1937**	1,372,459	4.1651	**1,370,985**	**4.1478**
200	0.9	41-42	46,668,747	237.377	**33,865,985**	**170.872**	12,841,102	69.204	**12,593,221**	**68.135**
300	0.4	9-10	11,532	0.0208	**9,472**	**0.018**	9,620	**0.018**	**9,473**	**0.018**
300	0.5	12	54,967	0.111	**45,867**	**0.093**	46,675	0.093	**45,760**	**0.089**
300	0.6	15	474,064	0.9898	**401,014**	**0.852**	**318,723**	0.738	323,228	**0.728**
300	0.7	20	6,008,385	15.747	**4,539,024**	**12.168**	3,567,911	9.515	**3,332,137**	**9.102**
500	0.3	8-9	18,993	0.0342	**16,002**	**0.033**	16,489	0.0357	**16,003**	**0.0343**
500	0.4	10-11	133,632	0.248	**121,231**	**0.233**	**119,799**	0.229	121,229	**0.222**
500	0.5	13-14	1,089,475	2.467	**950,809**	**2.1705**	**826,481**	**1.982**	828,042	1.983
500	0.6	17	15,643,333	42.476	**13,198,872**	**36.270**	10,540,503	28.786	**8,272,422**	**28.612**
1000	0.2	7	44,672	**0.0864**	**41,012**	0.095	41,134	0.095	**41,009**	0.095
1000	0.3	9-10	421,701	0.793	**389,710**	**0.754**	391,015	**0.757**	**389,706**	0.758
1000	0.4	12	4,312,069	10.581	**3,832,100**	**9.746**	3,268,019	**9.186**	**3,229,398**	9.251
3000	0.1	6-7	**140,415**	**0.422**	141,967	0.623	142,058	0.654	**141,966**	**0.626**
3000	0.2	9	2,926,645	**7.1825**	**2,685,401**	7.4582	2,715,672	**7.3828**	**2,685,396**	7.4708

TABLE 2. CPU times and number of steps (recursive procedure calls) for random graphs. The best times and minimum number of steps for each row are written in bold and underline style. Bold values in columns 4 and 5 indicate the best performance between MCQ+CS and NK-MaxClique and bold values in the last two columns indicate the priority of the MaxCliqueDyn or NK- MaxCliqueDyn algorithms.

5. **Conclusions and future works**

In this paper, we studied the Maximum Clique Problem (MCP), an important problem in graph theory and real-world applications. For MCP, we presented two new Branch-and-Bound exact algorithms, which employ a novel pruning bound (i.e. LCN_{NK}) to ignore the subproblems that can not contain larger clique(s) than the current solution. The other advantage of the algorithms is that they are designed to find a clique including vertex u in $NK(u)(\subseteq N(u))$, instead of $N(u)$. The computational results reveal that the new bound and smaller search area help to improve the MCQ and MaxCliqueDyn algorithms.

As a future work we will study the new pruning parameter and search area restriction on the well-known existing algorithms.

http://dx.doi.org/10.22108/toc.2021.120153.1686

Appendix A. The Core-matrix, NK-MaxCliqueDyn and Modified-MaxCliqueDyn Algorithms

In what follows we briefly describe the Core-matrix, NK-MaxCliqueDyn and Modified-MaxCliqueDyn algorithms. The Core-matrix algorithm provides matrix K and orders the vertices of a given graph. The NK-MaxCliqueDyn algorithm is presented in Algorithm 4. The Modified-MaxCliqueDyn algorithm is a new version of the MaxCliqueDyn algorithm which is modified with respect to the new pruning conditions 3.1 and 3.2. In the Modified-MaxCliqueDyn algorithm, $level$ denotes the number of branches from the root of the recursion tree to the current leaf. $S[level]$ and $S_{old}[level]$ denote the sum of steps which the Modified-MaxCliqueDyn algorithm performs from the root node up to and including the current level, and the sum of steps up to and including the previous level, respectively. $T[level] = S[level]/ALL\_STEPS$ where ALL_STEPS is a global counter of steps which is increased by 1 at each step of the Modified-MaxCliqueDyn algorithm. While $T[level] < T_{limit}$, the algorithm performs the calculations of the degrees and sorting. For more details see [19]. The formal description of the Modified-MaxCliqueDyn algorithm is depicted in Algorithm 5.

Algorithm 3 *The* Core-matrix(G) algorithm

Input: the graph $G = (V, E)$.
Output: the core-matrix K and the ordered vertices set $\bar{V}$.
Order the set of vertices V in increasing order of their degrees;
$i := 1$;
$\bar{V} = \emptyset$;
for $u \in V$ in the order **do**
  $core(u) := \deg(u)$;
  $order(u) := i$;
  $i := i + 1$;
  $\bar{V} := \bar{V} \cup \{u\}$;
  for $v \in N(u)$, where $v \notin \bar{V}$ **do**
    $K_{uv} := 1$;
    if $\deg(v) > \deg(u)$ **then**
      delete edge (u, v) from G and let $\deg(v) := \deg(v) - 1$;
      reorder V accordingly;
    end if
  end for
end for

http://dx.doi.org/10.22108/toc.2021.120153.1686

Algorithm 4 *T*he NK-MaxCliqueDyn(G) algorithm

Input: undirected graph $G = (V, E)$.
Output: A maximum clique Q_{max} of G.
$(K,\ \bar{V}) := core - matrix(G)$;
$Q_{max} := \emptyset$; for every $u \in \bar{V}$ compute $LCN(u)$ and $LCN_{NK}(u)$.
for every $u \in \bar{V}$ in the order **do**
  if $|Q_{max}| < LCN_{NK}(u) + 1$ **then**
    calculate the degrees of vertices in $G[NK(u)]$ and sort vertices in $NK(u)$ in a non-increasing order with respect to their degrees;
    $C := ColorSort(G[NK(u)])$;
    $Modified - MaxCliqueDyn(NK(u), LCN, C, 1)$;
  end if
end for
retun Q_{max};

http://dx.doi.org/10.22108/toc.2021.120153.1686

Algorithm 5 *The* Modified-MaxCliqueDyn$(V', LCN, C, level)$ algorithm

Input: vertex set $V' \subseteq V$,Local Core number set LCN, Color set C.
Output: A maximum clique of $G[V']$.
$S[level] := S[level] + S[level-1] - S_{old}[level]$;
$S_{old}[level] := S[level-1]$;
while $V' \neq \emptyset$ **do**
  Choose a vertex v with maximum color $C(v)$ from V'.
  $V' := V' \setminus \{v\}$;
  if $LCN(v) + 1 > |Q_{max}|$ and $|Q| + C(v) > |Q_{max}|$ **then**
    $Q := Q \cup \{v\}$;
    if $V' \cap N(v) \neq \emptyset$ and $|V' \cap N(v)| + |Q| > |Q_{max}|$ **then**
      if $S[level]/ALL\_STEPS < T_{limit}$ **then**
        calculate the degrees of vertices in $G[V' \cap N(v)]$;
        sort vertices in $V' \cap N(v)$ in a non-increasing order with respect to their degrees;
      end if
      $C^{'} = ColorSort(G[V' \cap N(v)])$;
      $S[level] := S[level] + 1$;
      $ALL\_STEPS := ALL\_STEPS + 1$;
      Modified-MaxCliqueDyn$(V' \cap N(v), LCN, C^{'}, level + 1)$;
    else
      if $(|Q| > |Q_{max}|)$ **then**
        $Q_{max} := Q$;
      end if
    end if
    $Q := Q \setminus \{v\}$;
  else
    return;
  end if
end while

References

[1] P. San Segundo and J. Artieda, A novel clique formulation for the visual feature matching problem, *Appl. Intell.*, **43** (2015) 325–342.

[2] P. San Segundo, D. Rodriguez-Losada, F. Matia and R. Galan, Fast exact feature based data correspondence search with an efficient bit-parallel MCP solver, *Appl. Intell.*, **32** (2010) 311–329.

[3] S. Butenko and W. E. Wilhelm, Clique-detection models in computational biochemistry and genomics, *European J. Operational Researc*, **173** (2006) 1–17.

[4] C. W. Art, B. Sergiy and P. Panos M., *Clustering challenges in biological networks*, World Scientific Publishing Co, Inc., 2009.

[5] T. Etzion and P. R. Ostergard, Greedy and heuristic algorithms for codes and colorings, *IEEE Trans. Inform. Theor*, **44** (1998) 382–388.

http://dx.doi.org/10.22108/toc.2021.120153.1686

[6] T. Gschwind, S. Irnich, F. Furini and R. W. Calvo, *Social network analysis and community detection by decomposing a graph into relaxed cliques*, Technical Report LM-2015-07; Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz: Mainz, Germany, 2015.

[7] O. Weide, D. Ryan and M. Ehrgott, An iterative approach to to robust and integrated aircraft routing and crew scheduling, *Comput. Oper. Res.*, **37** (2010) 833–844.

[8] Q. Wu and J. K. Hao, A review on algorithms for maximum clique problems, *European J. Oper. Res.h*, **242** (2015) 693–709.

[9] V. Batagelj and M. Zaversnik, $O(m)$ *algorithm for cores decomposition of networks*, arXiv preprint cs/0310049, 2003 1–9.

[10] S. B. Seidman, Network structure and minimum degree, *Social Networks*, **5** (1983) 269–287.

[11] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, A Series of Books in the Mathematical Sciences. W. H. Freeman and Co., San Francisco, Calif., 1979.

[12] J. Hastad, Clique is hard to approximate within $n^{1-\epsilon}$, *Acta Math.*, **182** (1999) 105–142.

[13] U. Feige, Approximating maximum clique by removing subgraphs, *SIAM J. Discrete Math.*, **18** (2004) 219–225.

[14] P. R. J. Ostergard, A fast algorithm for the maximum clique problem, *Discrete Appl. Math.*, **120** (2002) 197–207.

[15] L. Babel and G. Tinhofer, A branch and bound algorithm for the maximum clique problem, *Z. Oper. Res.*, **34** (1990) 207–217.

[16] D. Brelaz, New methods to color the vertices of a graph, *Comm. ACM*, **22** (1979) 251–256.

[17] E. Tomita and T. Seki, An efficient branch-and-bound algorithm for finding a maximum clique. *In:Proceedings of the discrete mathematics and theoretical computer science. Lecture Notes in Computer Science*, **2731** (2003) 278–289.

[18] E. Tomita and T. Kameda, An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments, *J. Global Optim.*, **37** (2007) 95–111.

[19] J. Konc and D. Janezic, An improved branch and bound algorithm for the maximum clique problem, *MATCH Commun. Math. Comput. Chem.*, **58** (2007) 569–590.

[20] P. San Segundo, D. Rodrguez-Losada and A. Jimenez, An exact bit-parallel algorithm for the maximum clique problem, *Comput. Oper. Res.*, **38** (2011) 571–581

[21] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi and M. Wakatsuki, A simple and faster branch-and bound algorithm for finding a maximum clique, *Lecture Notes in Comput. Sci.*, **5942** (2010) 191–203.

[22] P. San Segundo, F. Matia, D. Rodriguez-Losada and M. Hernando, An improved bit parallel exact maximum clique algorithm, *Optim. Lett.*, **7** (2013) 467–479.

[23] P. San Segundo, C. Tapia, Relaxed approximate coloring in exact maximum clique search, *Comput. Oper. Res.*, **44** (2014) 185–192.

[24] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin and M. M. A. Patwary, Fast maximum clique algorithms for large graphs, *In: Proceedings of the 23rd International Conference on World wide web, ACM*, (2014) 365–366.

[25] R. A. Rossi, D. F. Gleich and A. H. Gebremedhin, Parallel maximum clique algorithms with applications to network analysis, *SIAM J. Sci. Comput.*, **37** (2015) C589–C616.

[26] B. Pattabiraman, M. M. A. Patwary, A. H. Gebremedhin, W. K. Liao and A. Choudhary, *Fast algorithms for the maximum clique problem on massive sparse graphs*, Algorithms and models for the web graph, Lecture Notes in Comput. Sci., **8305**, Springer, Cham, (2013) 156-169

[27] P. San Segundo, A. Lopez and P. M. Pardalos, A new exact maximum clique algorithm for large and massive sparse graphs, *Comput. Oper. Res.*, **66** (2016) 81–94.

[28] P. San Segundo, A. Nikolaev and M. Batsyn, Infra-chromatic bound for exact maximum clique search, *Comput. Oper. Res.*, **64** (2015) 293–303.

[29] C. M. Li and Z. Quan, An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem, *In: Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI-10*, (2010) 128–133.

http://dx.doi.org/10.22108/toc.2021.120153.1686

[30] D. J. Welsh and M. B. Powell, An upper bound for the chromatic number of a graph and its application to timetabling problems, *Comput. J.*, **10** (1967) 85–86.

[31] J. T. Hungerford and F. Rinaldi, A General Regularized Continuous Formulation for the Maximum Clique Problem, *Math. Oper. Res.*, **44** (2019) 1161–1173.

[32] K. Kanazawa and S. Cai, FPGA Acceleration to Solve Maximum Clique Problems Encoded into Partial MaxSAT, *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, (2018) 217–224.

[33] M. T. Belachew and N. Gillis, Solving the Maximum Clique Problem with Symmetric Rank-One Non-negative Matrix Approximation, *J. Optim. Theory Appl.*, **173** (2017) 279–296.

[34] A. Verma, A. Buchanan and S. Butenko, Solving the maximum clique and vertex coloring problems on very large sparse networks, *INFORMS J. Comput.*, **27** (2015) 164–177.

[35] C. M. Li, Z. Fang, H. Jiang and K. Xu, Incremental upper bound for the maximum clique problem, *INFORMS J. Comput.*, **30** (2017) 137–153.

Neda Mohammadi

Department of Computer Science, University of Shahrekord, Shahrekord, Iran

Email: neda.m@stu.sku.ac.ir

Mehdi Kadivar

Department of computer science, University of Shahrekord, Shahrekord, Iran

Email: m_kadivar@aut.ac.ir

http://dx.doi.org/10.22108/toc.2021.120153.1686