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 a 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 its 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 $\text{deg}(v)$. 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, in computer vision [1],

*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 BBMCS for large and massive sparse graphs using a novel sparse encoding for the adjacency matrix. Like PMC, in BBMCS core
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 $\text{core}(u)$, 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 $\text{core}(v) \geq \text{core}(u)$.
Proposition 2.2. The core number of elements in $NK(u)$ is at least $\text{core}(u)$. Therefore, the members in $NK(u)$ have the necessary condition (i.e. the core number of each member must be at least $\text{core}(u)$) to construct a clique of size $\text{core}(u) + 1$ (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 $\text{LCN}(u)$. 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 $\text{LCN}_{NK}(u)$.

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

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 $Q_{\text{max}}$ be a maximum clique in $G$. Then, $Q_{\text{max}} \subseteq NK(u)$ in which $u$ is the first vertex among elements in $Q_{\text{max}}$ 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 $\tilde{V} = (v_{10}, v_{11}, v_9, v_8, v_7, v_6, v_5, v_4, v_3, v_2, v_1)$ and the $K$ matrix is computed as follow:

\begin{verbatim}
  v1  v2  v3  v4  v5  v6  v7  v8  v9  v10  v11
v1  1   1   1   0   1   1   1   0   0   0   0
v2  1   1   1   1   1   1   1   1   0   0   0
v3  1   1   1   1   1   1   1   1   1   0   0
v4  0   1   1   1   1   1   1   1   1   0   0
v5  0   0   1   1   1   1   1   1   0   0   0
v6  0   0   0   1   1   1   1   1   0   0   0
v7  0   0   0   1   1   1   1   1   0   0   0
v8  1   0   0   0   1   1   1   1   0   0   0
v9  0   1   0   0   0   1   1   1   0   0   0
v10 0   0   0   0   1   1   1   1   0   0   0
v11 0   0   0   0   0   1   1   1   0   0   0
\end{verbatim}

Figure 1. Graph $G$ in Example 1.

http://dx.doi.org/10.22108/toc.2021.120153.1686
respectively. By the definition of core number, we can see that

Proposition 2.6. That its core number is computed by the Core-matrix algorithm.

As the first vertex, $v_{10}$ is selected and then $\text{core}(v_{10}) = 1$ 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 $\text{core}(v_{11}) = 1$ 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:

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

$$\text{core}(v_8) = \text{core}(v_1) = \text{core}(v_5) = \text{core}(v_4) = 2,$$

$$\text{core}(v_2) = \text{core}(v_6) = \text{core}(v_7) = \text{core}(v_3) = 3.$$

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 $Q_{\text{max}} \subseteq NK(u)$. Therefore, instead of $N(u)$, our algorithm searches in $NK(u)$ to find $Q_{\text{max}}$. 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 $V = \{v_1, v_2, \ldots, v_n\}$ 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 $u \in V$ and $Q^u_{\text{max}}$ and $Q^u_{NK-\text{max}}$ are maximum cliques in $G[N[u]]$ and $G[NK(u)]$, respectively. By the definition of core number, we can see that $|Q^u_{\text{max}}| \leq LCN(u) + 1 \leq \text{core}(u) + 1$ and

$$\text{core}(u) = \max\{\text{core}(v) \mid v \in V \text{ and } (v, u) \in E\}.$$
\[ |Q_{NK_{-max}}^u| \leq LCN_{NK}(u) + 1 \leq LCN(u) + 1 \leq core(u) + 1. \] Let \( LCN_{\text{max}} = \max \{ LCN_{NK}(u) \mid u \in V \}. \) We have \( \omega(G) \leq LCN_{\text{max}} + 1. \)

### 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 considered 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 \( Q_{\text{max}} \) be the set of vertices of the largest clique found until now. Initially, the NK-MaxClique algorithm puts \( Q_{\text{max}} = \emptyset \). 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 \( |Q_{\text{max}}| \geq LCN_{NK}(u) + 1 \) holds, it is implied that there is no clique larger than \( Q_{\text{max}} \) 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

1: Input: undirected graph \( G = (V, E) \).
2: Output: A maximum clique \( Q_{\text{max}} \) of \( G \).
3: \((K, \bar{V}) := \text{core}\text{-matrix}(G)\);
4: \(Q_{\text{max}} := \emptyset\); for every \( u \in \bar{V} \) compute \( LCN(u) \) and \( LCN_{NK}(u) \).
5: for every \( u \in \bar{V} \) in the order do
6: if \( |Q_{\text{max}}| < LCN_{NK}(u) + 1 \) then
7: 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;
8: \( C := \text{ColorSort}(G[NK(u)]) \);
9: \( \text{Modified - MCQ}(NK(u), LCN, C) \);
10: end if
11: end for
12: return \( Q_{\text{max}} \);

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 another upper bound for the maximum clique including $v$. Therefore, if $|Q_{\text{max}}| < |Q| + C(v)$ and $|Q_{\text{max}}| < LCN(v) + 1$, $Q$ 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

\begin{equation}
LCN(v) + 1 > |Q_{\text{max}}|
\end{equation}

and

\begin{equation}
|V' \cap N(v)| + |Q| > |Q_{\text{max}}|
\end{equation}

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 $V' \subseteq V$, Local Core number set $LCN$, Color set $C$.
2: Output: A maximum clique of $G[V']$.
3: while $V' \neq \emptyset$ do
4: Choose a vertex $v$ with maximum color $C(v)$ from $V'$.
5: $V' := V' \setminus \{v\}$;
6: if $LCN(v) + 1 > |Q_{\text{max}}|$ and $|Q| + C(v) > |Q_{\text{max}}|$ then
7: $Q := Q \cup \{v\}$;
8: if $V' \cap N(v) \neq \emptyset$ and $|V' \cap N(v)| + |Q| > |Q_{\text{max}}|$ then
9: $C' = \text{ColorSort}(G[V' \cap N(v)])$;
10: Modified-MCQ($V' \cap N(v)$, $LCN$, $C'$);
11: else
12: if $(|Q| > |Q_{\text{max}}|)$ then
13: $Q_{\text{max}} := Q$;
14: end if
15: end if
16: $Q := Q \setminus \{v\}$;
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).
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.
Table 1. CPU times and number of steps (recursive procedure calls) for DIMACS benchmark graphs. \( \omega \) is the clique number, \( \Delta G \) is the maximum degree. \( LCN_{\text{max}} \) and \( core_{\text{max}} \) 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
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.

\[
\text{http://dx.doi.org/10.22108/toc.2021.120153.1686}
\]
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

1: Input: the graph $G = (V,E)$.
2: Output: the core-matrix $K$ and the ordered vertices set $\bar{V}$.
3: Order the set of vertices $V$ in increasing order of their degrees;
4: $i := 1$;
5: $\bar{V} = \emptyset$;  
6: for $u \in V$ in the order do 
7: \hspace{1cm} $core(u) := \text{deg}(u)$;
8: \hspace{1cm} $order(u) := i$;
9: \hspace{1cm} $i := i + 1$;
10: \hspace{1cm} $\bar{V} := \bar{V} \cup \{u\}$;
11: end for
12: for $v \in N(u)$, where $v \notin \bar{V}$ do
13: \hspace{1cm} $K_{uv} := 1$;
14: \hspace{1cm} if $\text{deg}(v) > \text{deg}(u)$ then 
15: \hspace{2cm} delete edge $(u,v)$ from $G$ and let $\text{deg}(v) := \text{deg}(v) - 1$;
16: \hspace{2cm} reorder $V$ accordingly;
17: \hspace{1cm} end if 
18: end for
19: end for
Algorithm 4 The NK-MaxCliqueDyn($G$) algorithm

1: Input: undirected graph $G = (V,E)$.
2: Output: A maximum clique $Q_{max}$ of $G$.
3: $(K, \bar{V}) := core - matrix(G)$;
4: $Q_{max} := \emptyset$; for every $u \in \bar{V}$ compute $LCN(u)$ and $LCN_{NK}(u)$.
5: for every $u \in \bar{V}$ in the order do
6: if $|Q_{max}| < LCN_{NK}(u) + 1$ then
7: 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;
8: $C := ColorSort(G[NK(u)])$;
9: $Modified - MaxCliqueDyn(NK(u), LCN, C, 1)$;
10: end if
11: end for
12: return $Q_{max}$;

http://dx.doi.org/10.22108/toc.2021.120153.1686
Algorithm 5 The Modified-MaxCliqueDyn($V', LCN, C, level$) algorithm

1: Input: vertex set $V' \subseteq V$, Local Core number set $LCN$, Color set $C$.
2: Output: A maximum clique of $G[V']$.
4: $S_{old}[level] := S[level - 1]$;
5: while $V' \neq \emptyset$ do
6: Choose a vertex $v$ with maximum color $C(v)$ from $V'$.
7: $V' := V' \setminus \{v\}$;
8: if $LCN(v) + 1 > |Q_{\text{max}}|$ and $|Q| + C(v) > |Q_{\text{max}}|$ then
9: $Q := Q \cup \{v\}$;
10: if $V' \cap N(v) \neq \emptyset$ and $|V' \cap N(v)| + |Q| > |Q_{\text{max}}|$ then
11: if $S[level]/ALL\_STEPS < T_{\text{limit}}$ then
12: calculate the degrees of vertices in $G[V' \cap N(v)]$;
13: sort vertices in $V' \cap N(v)$ in a non-increasing order with respect to their degrees;
14: end if
15: $C' = \text{ColorSort}(G[V' \cap N(v)])$;
16: $S[level] := S[level] + 1$;
17: $ALL\_STEPS := ALL\_STEPS + 1$;
18: Modified-MaxCliqueDyn($V' \cap N(v), LCN, C', level + 1$);
19: else
20: if $(|Q| > |Q_{\text{max}}|)$ then
21: $Q_{\text{max}} := Q$;
22: end if
23: end if
24: $Q := Q \setminus \{v\}$;
25: else
26: return;
27: end if
28: end while

References


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


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