Testing a Heuristic Algorithm for Finding a Maximum Clique on DIMACS and Facebook Graphs

VLADIMIR BALASH
Saratov State University
Faculty of Mechanics and Mathematics
83 Astrakhanskaya Street
RUSSIAN FEDERATION
vladimirbalash@yandex.ru

ANASTASIA STEPANOVA
Saratov State University
Risk Institute
83 Astrakhanskaya Street
RUSSIAN FEDERATION
stacey.stepanova@gmail.com

DANIIL VOLKOV
Saratov State University
Faculty of Computer Science and IT
83 Astrakhanskaya Street
RUSSIAN FEDERATION
volkovda351@gmail.com

SERGEI MIRONOV
Saratov State University
Faculty of Computer Science and IT
83 Astrakhanskaya Street
RUSSIAN FEDERATION
mironovsv@info.sgu.ru

ALEXEY FAIZLIEV
Saratov State University
Risk Institute
83 Astrakhanskaya Street
RUSSIAN FEDERATION
faizlievar1983@mail.ru

SERGEI SIDOROV
Saratov State University
Risk Institute
83 Astrakhanskaya Street
RUSSIAN FEDERATION
sidorovsp@yahoo.com

Abstract: In this paper we propose a new heuristic algorithm for solving a maximum clique search problem (MCP). While the proposed algorithm (called *TrustCLQ*) uses a general approach to solving MCP, it is almost independent of the order of vertices and does not exploit a partition of the graph into independent sets. The algorithm was tested on DIMACS library graphs which are often employed for testing MCP solution algorithms. *TrustCLQ* algorithm was compared with the well-known ILS heuristic algorithm (as well as with a standard algorithm from *networkx* library) on DIMACS data sets. Moreover, *TrustCLQ* algorithm has been tested on Facebook social graphs.

Key–Words: Heuristic algorithm, Graphs, Cliques, Network analysis, Maximum clique problem, DIMACS graphs, Social graphs

Received: April 3, 2020. Revised: April 27, 2020. Accepted: April 30, 2020. Published: April 30, 2020.

1 Introduction

The Maximum Clique Problem (MCP) is one of the most well-known NP-hard problems in graph theory [1]. A clique in a graph is a subset of vertices that form a complete subgraph. A maximum clique in a graph is the maximum-sized subset of such vertices.

Applications of the maximum clique problem are quite wide. In bioinformatics, MCP is used in computer analysis of genomic databases, e.g. in the search for potential regulatory structures of ribonucleic acids. In social networks, MCP is used for clustering data, e.g. for dividing various communities into groups (clusters) that share common properties. Cluster allocation allows each of them to be processed by a separate auxiliary server. In chemistry, the MCP problem underlies the search for the maximum general substructure in the graph describing the structure of the chemical compound. In addition, MCP is a mathematical model of a number of problems arising in the electronic equipment design.

In these applications, exact MCP solutions are usually required. However, the size of input data is usually too huge (input graphs can contain up to a million vertices). Thus, the important topic of MCP research is the development of new approaches for finding MCP solutions taking into account the features of

graphs that arise in applications.

Let $G = (V, E)$ be an undirected graph, where $V = \{1, \ldots, n\}$ is the set of vertices and $E \subset V \times V$ is the set of edges. Let $G(S) = (S, E \cap S \times S)$ denote the subgraph formed by the subset of vertices S. The notion $\overline{G}$ will be used for the complement of the graph G, $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{(i, j) | i, j \in V, i \neq j \text{ and } (i, j) \notin E\}$. Two vertices i and j are adjacent if $(i, j) \in E$. The neighbors of vertex i are the set of vertices adjacent to i, $N(i) = \{j \in V : (i, j) \in E\}$. Let A_G denote the adjacency matrix of graph G; the adjacency matrix has size $n \times n$, $A_G = (a_{ij})$, where $a_{ij} = 1$ if $(i, j) \in E$ and $a_{ij} = 0$, otherwise. The degree $deg(i)$ of vertex i in G is defined as the number of neighbors of vertex i in G and will be denoted by $|N(i)|$. Let $\delta(G)$ and $\Delta(G)$ denote the least and greatest degree of vertices in G, respectively.

A graph $G = (V, E)$ is called complete if all its vertices are adjacent to each other, i.e. if $\forall i, j \in V$, $i \neq j$, $(i, j) \in E$. We define a clique C as a subset of vertices such that $G(C)$ is complete. An independent set in G is a subset of the vertices of I such that the subgraph $G(I)$ formed by them does not contain edges. A clique (independent set) is called maximum if there is no larger clique (independent set) in the graph.

The clique number is the number of vertices in the maximum clique in G and traditionally denoted by $\omega(G)$. The independence number $\alpha(G)$ is defined similarly. It is easy to see that C is a maximum clique in G if and only if C is a maximal independent set in $\overline{G}$, therefore $\omega(G) = \alpha(\overline{G})$.

There are several dozens of algorithms for solving MCP which can be classified either exact algorithms or heuristic ones. Exact algorithms are usually based on exhaustive search in conjunction with the branch-and-bound method or different vertex-coloring schemes [2]. The exact algorithms such as the MCQ algorithm [3], MCR [4], MCS [5], MaxCliqueDyn [6], MaxCliquePara (MCP) [7], BBmaxClique [8], FastMaxClique (FMC) [9], parallel maximum clique (PMC) algorithm [10], an implementation of a MotzkinStraus-based iterative clique-finding algorithm for GPUs [11], among many others, have proved their applicability in solving MCP. One of the currently best exact approaches was proposed in the paper [12].

Heuristic algorithms employ a wide variety of ideas including reactive local search [13], randomness in DLS [14] and CLS algorithms [15], a deterministic greedy heuristic that adds vertices in an order depending on their weights in UALEX-MS algorithm [16], a hybrid evolutionary method in EAG [17]. Often, specific properties of the studied graph are used, e.g. a special order of vertices, the known lower and upper bounds, the chromatic number of a graph, etc. Exact algorithms are ineffective in solving NP-complete problems, since they are based on exhaustive search. While heuristic algorithms are not accurate, they may be more promising in real applications, since they obtain the solution in much lesser time. Heuristic algorithms all differ in accuracy, speed, and they often use specific properties of graphs. One of the currently best heuristic methods is LSCC+BMS, proposed in the paper [18].

The recent development on MCP can be found in papers [19, 20, 21, 22, 23, 24, 25, 26, 27, 28].

In this paper, one of such algorithms is proposed and examined. We present a novel simple iterative algorithm for maximum clique finding. The standard iterative approach is extended by using the trust indicator of vertices. The proposed method belongs to the class of heuristic optimization methods. Therefore, the proposed algorithm may be convenient for solving many real-world problems related to large social networks.

In this paper, the proposed algorithm is compared with the ILS heuristic algorithm [29] as well as with a standard algorithm implemented in the `networkx` library `https://networkx.github.io` (`max_clique()`, below `nx.max_clique` functions) on DIMACS data sets. The empirical results show that the TrustCLQ algorithm performs better than the standard algorithm from `networkx` library. Moreover, `TrustCLQ` algorithm has been tested on Facebook social graphs.

2 Trust-Based Algorithm (TrustCLQ)

The main ideas used in heuristic algorithms include vertex cut-off, independent set cut-off, centralization coefficient selection, pseudo-random selection. The proposed algorithm is based on some of these ideas. An effective fast heuristic should combine as many strengths as possible (including accuracy) and as few weaknesses as possible (e.g. execution time). Moreover, the rejection of one or another approach should provide clear advantages. It is not possible to obtain high accuracy without significant loss of time, because one way or another it all comes down to a complete enumeration of all branches of the decision tree.

2.1 Vertex trust indicator

The new approach in the MCP solution proposed in this paper will be based on the use of the trust indicator for each vertex in the graph. It has the following intuitive practical meaning. If we expected that the vertex v will form a clique of size k, but it has not happened at an iteration, then we decrease confidence in the vertex. Since vertices with a greater degree are less likely to form a clique of size k on the first attempt, the trust in them should melt away more slowly. In this paper we propose to evaluate the vertex trust indicator by

$$trust(v) = trust(v) - \frac{k}{deg(v)+1} \qquad (1)$$

at each iteration of the algorithm.

Note that the vertex trust indicator may also incorporate a dependence on the binomial coefficient, since the maximum number of cliques of size k in a graph with n vertices is equal to $\frac{n!}{k!(n-k)!}$. However, due to the sparseness of real graphs, the number of possible cliques is much less than expected. For this reason, we use a very simplified and approximate method of evaluating a vertex trust defined by equation (1). As will be shown below, this approach showed acceptable accuracy on sparse graphs.

The initial value of the trust indicator of each vertex can be set to 1. This indicator can be changed before running the algorithm. The higher the value of the trust indicator for a vertex, the higher should be the accuracy of the solution obtained by the algorithm, since a larger value of the trust indicator increases the number of iterations, each of which has a chance to improve the solution.

This method allows anyone to search for maximum cliques in graphs about which nothing is known in advance. The algorithm does not use the concepts of graph coloring and the number of its independent sets. In this sense, the proposed approach is more universal. At the same time, one can adjust the accuracy of the algorithm by changing the initial value of vertex trust indicator. If one needs to get an approximate solution on a large graph, then the initial value of vertex trust indicator can be set to minimum. If one need to obtain the most possible accurate solution, then the indicator should be set higher.

Obviously, if there is a clique of size k in the graph, then there is also a clique of size $k-1$. It would be logical that one should not continue to search for large cliques in the absence of smaller ones. However, this is only correct with respect to exact algorithms.

When it comes to our heuristic algorithm, then equation (1) comes into play to find an heuristic evaluation of each vertex. As one can see, with an increase in the size of the clique k, the number of possible cliques decreases. Therefore, by cutting off vertices with a small degree, such algorithms are assumed to be able to demonstrate a good accuracy. Thus, the probability of finding the best solution may be increasing.

2.2 Vertex removal

One of the elements of our algorithm is the removal of irrelevant vertices. It may work in a positive way, since it reduces the size of the search tree and increases the likelihood of finding the optimal solution. In our algorithm, we will remove a vertex in two cases.

- In the first case, the vertices are removed using the degree-based criterion. Suppose we are looking for a clique of size k in a graph. Then all vertices with degree $deg(v) + 1 < k$ should be permanently removed from the graph. This will not affect the solution, but only allow it to be improved due to the fact that the "useless" vertices will not affect the size of the search tree.

- In the second case, the vertices are removed when a confidence in the vertex is lost. If the trust indicator (1) of vertex v at a certain iteration becomes less than zero, i.e. $trust(v) < 0$, then the vertex is removed from the graph.

2.3 TrustCLQ algorithm

The proposed algorithm (called `TrustCLQ` algorithm) consists of several functions. The main function `findMaxClique` accepts an undirected and unweighted graph without multi-edges and loops. Vertex numeration starts from zero. If nothing is known about the upper border of the clique, then we can safely assume that it is equal to the number of vertices in the graph. If the lower boundary is unknown, then it is equal to 1 or $\delta(G)$, i.e. the least degree of vertices in G. The initial solution is the empty set. The pseudocode of `findMaxClique` function can be found in Algorithm 1. The function moves from the bottom boundary to the top boundary. If a clique of size i is searched, then the function immediately removes from the graph all vertices with degree $deg(v)+1 < i$. They will no longer be considered in the algorithm execution. Vertices from the graph are removed using the `reduceVertex` function (Algorithm 4).

Then the main function `findMaxClique` runs the clique search function named `findClique` on the updated graph and update our solution. The pseudocode of function `findClique` is presented in Algorithm 2. The function directly chooses a solution and calculates the trust indicator (1) for each vertex. The global variable `SCALE` is responsible for that and its value can be changed before each run of the algorithm.

Function `findClique` takes the next vertex and tries to build a solution that will be no worse than the already known solution, i.e. it should be a clique of size at least LB. For each vertex, a list of candidates is formed that can be included in the solution. The Bron-Kerbosch algorithm is employed to perform a complete search of all possible variations of candidates. At each iteration, `TrustCLQ` selects one of them randomly. This candidate is added to the solution without any checks, since the meaning of the list of candidates is that they are all suitable for inclusion to the solution. After that, the list is updated to the current one.

The update function `updateCandidates` is presented in Algorithm 3. While this reduces the search tree to a minimum, it reduces the accuracy. If the solution has been improved, then control is transferred back to function `findMaxClique`.

An important feature of the algorithm is that it allows to control the accuracy by changing the initial value of the trust indicator (the SCALE parameter). For example, with the initial value of the trust indicator of $\frac{1}{n}$, where n is the number of vertices in the graph, the algorithm has low accuracy. However, with an increase of its value, the accuracy increases (of course, along with the running time of the algorithm). One can choose between a high initial value of the trust indicator, which leads to a higher running time, but provides a solution close to optimal, or a low initial value of the trust indicator, which gives a quick but an inaccurate result.

Note that the algorithm does not require a significant amount of additional memory to work. It stores only a local copy of the graph and several sets whose cardinality does not exceed the number of the graph vertices.

The running time is difficult to estimate, since it also depends on the internal structure of the graph. In the course of the empirical study, it could be noted that the average value of the vertex degree has a great influence on the running time. Indeed, despite the fact that there are a lot of links in social graphs, they are processed by the algorithm quite quickly, since their density is very low.

The algorithm can also be used to solve the problem of an independent set finding. In this case, it is necessary to take the complement of the graph and submit it as the input to the algorithm.

Algorithm 1 $findMaxClique(G)$

```
answer ← ∅
for i = 1; i < |G| do
    for all v ∈ G do
        if |N(v)| + 1 < i then
            reduceVertex(G, v)
        end if
    end for
    tmp ← findClique(G, |answer| + 1)
    if |tmp| > |answer| then
        answer ← tmp
        i ← |answer| + 1
    else
        i ← i + 1
    end if
end for
return answer
```

If the solution has not been improved at an iteration, then the trust indicator of the vertex, from which this solution was formed, decreases according to equation (1). If the value of the trust indicator of the vertex becomes negative, then the vertex is removed from the graph. If the algorithm does not find a clique that improves the solution, then it returns the empty set. However, this does not unambiguously indicate that a maximum clique has been found. The solution can still be improved by cutting off another vertices and restarting the algorithm.

The algorithm was implemented in Python and its code can be found by link `https://github.com/stacy-s/MCP`. All runs were carried out on the Amazon EC, Intel Xeon Platinum 8000 series (Skylake-SP) 3.1 GHz, 2vCPU, 1Gb.

3 Empirical results

3.1 DIMACS graphs

To test TrustCLQ algorithm, we will use several graphs from DIMACS benchmark set `http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark`, which are usually employed to test the algorithms designed to solve the maximum clique problem. All

Algorithm 2 $findClique(G, LB)$

for all $v \in G$ **do**
  $trust_v \leftarrow SCALE$
end for
while there is at least one vertex with non-negative trust indicator $trust_v$ **do**
  for all $v \in G$ **do**
    if $trust_v > 0$ **then**
      $candidates \leftarrow N(v)$
      $clique \leftarrow \{v\}$
      while $candidates$ is not empty and $|clique| + |candidates| \geq LB$ **do**
        $u \leftarrow$ a vertex randomly chosen from $candidates$
        $clique \leftarrow clique \cup \{u\}$
        $updateCandidates(G, candidates, u)$
      end while
      if $|clique| \geq LB$ **then**
        return $clique$
      else
        $trust_v \leftarrow trust_v - \frac{LB}{|N(v)+1|}$
        if $trust_v \leq 0$ **then**
          $reduceVertex(g, v)$
        end if
      end if
    end if
  end for
end while
return $\varnothing$

graphs differ from each other in their characteristics. The DIMACS benchmark set consists of the graphs that are widely used to test the solution for MCP [30]. All test graphs are constructed based on various principles. For example, graphs of the `gen` family are generated as follows: they take as a basis a complete subgraph of a known size and add the necessary number of edges to it.

Table 1 shows the results and running time for the `TrustCLQ`, `nx.max_clique` algorithms as well as the results for the ILS algorithm (results for the ILS algorithm were obtained in [31]).

The algorithm showed the most inaccurate results when using the initial trust indicator value of 0.01. It can be explained by the fact that almost always after an unsuccessful construction of a clique, the vertex is removed from the graph, since the value of the trust indicator immediately becomes negative.

Algorithm 3 $updateCandidates(G, candidates, u)$

$tmp \leftarrow \varnothing$
for all $v \in candidates$ **do**
  if there is an edge (u, v) **then**
    $tmp \leftarrow tmp \cup \{v\}$
  end if
end for
return tmp

Algorithm 4 $reduceVertex(G, v)$

remove v from V (list of vertices)
for all $v \in E$ (E – list of all edges) **do**
  if there is an edge (u, v) or (v, u) **then**
    remove this edge from E
  end if
end for

Empirical results presented in Table 1 shows that the `TrustCLQ` algorithm running time and its accuracy is increasing with an increase in the value of the SCALE parameter. The results also show that the `TrustCLQ` algorithm has a greater accuracy and a shorter running time than the `nx.max_clique` algorithm on almost all graphs from the DIMACS data set. Moreover, in some cases, the `TrustCLQ` algorithm obtains the exact solution and performs no worse than the ILS algorithm.

3.2 The performance of the TrustCLQ algorithm on Social networks

The portal [32] contains several hundred of different real networks arisen in a wide range of applied areas including biological networks, chemical structures, economic relationships, infrastructure and road networks, and social networks. The TrustCLQ algorithm was tested on dozens of Facebook graphs taken from [32].

These graphs are very different from DIMACS graphs in terms of the node degree distribution as well as the edge density. Table 2 presents some of the obtained results. It shows that even with a large number of nodes and edges, the algorithm works much faster than on DIMACS graphs. This is due to the much greater sparseness of social networks, since a persons ability to form social connections with other network participants is limited. In addition, in the social networks, the spread in the vertex degrees is greater than in random graphs.

	Dataset	$\|V\|$	$t_{0.1}$	$\omega_{0.1}$	$t_{1.0}$	$\omega_{1.0}$	ω_{ILS}	ω_{best}	t_{nx}	ω_{nx}
0	C1000.9	1000	185.63	52	2015.27	54	-	-	149.5	47
1	C125.9	125	0.05	30	0.49	31	-	-	0.63	26
2	C2000.5	2000	322.74	14	2705.67	14	-	-	-	-
3	C250.9	250	0.38	38	11.37	39	-	-	3.52	33
4	C500.9	500	4.24	44	89.23	47	-	-	21.9	40
5	DSJC1000_5	1000	19.01	14	157.96	14	-	-	74.12	11
6	DSJC500_5	500	1.28	11	11.66	12	-	-	10.83	10
7	MANN_a27	378	8.27	123	61.86	123	126	126	10.66	104
8	MANN_a45	1035	507.19	335	6154.56	337	344	345	178.72	276
9	MANN_a9	45	0.0	15	0.01	16	16	16	0.05	13
10	brock200_1	200	0.08	18	1.54	19	21	21	1.21	16
11	brock200_2	200	0.04	9	0.56	12	12	12	0.96	8
12	brock200_3	200	0.05	14	0.72	13	15	15	1.07	11
13	brock200_4	200	0.07	13	0.82	14	17	17	1.15	12
14	brock400_1	400	2.07	21	26.44	22	23	27	7.74	18
15	brock400_2	400	2.78	20	19.19	21	24	29	8.05	18
16	brock400_3	400	1.88	20	31.33	22	24	31	7.56	19
17	brock400_4	400	2.96	21	19.03	22	26	33	7.59	18
18	brock800_1	800	16.81	17	181.63	18	19	23	44.56	15
19	brock800_2	800	17.41	17	175.39	18	20	24	45.0	16
20	brock800_3	800	20.56	18	162.28	18	19	25	44.15	16
21	brock800_4	800	15.47	17	173.13	18	19	26	44.15	16
22	c-fat200-1	200	0.01	12	0.01	12	12	12	2.56	12
23	c-fat200-2	200	0.01	24	0.02	24	24	24	2.11	24
24	c-fat200-5	200	0.03	58	0.05	58	58	58	1.98	58
25	c-fat500-1	500	0.06	14	0.07	14	14	14	43.89	14
26	c-fat500-10	500	0.29	126	0.55	126	126	126	22.49	126
27	c-fat500-2	500	0.08	26	0.09	26	26	26	39.65	26
28	c-fat500-5	500	0.15	64	0.22	64	64	64	29.28	64
29	gen200_p0.9_44	200	0.2	33	2.45	35	44	44	1.51	30
30	gen200_p0.9_55	200	0.19	34	2.43	42	55	55	1.76	37
31	gen400_p0.9_55	400	1.63	40	69.33	45	54	55	8.35	37
32	gen400_p0.9_65	400	1.7	41	60.5	45	64	65	9.11	38
33	gen400_p0.9_75	400	1.65	41	34.1	45	75	75	10.27	40
34	hamming10-2	1024	176.39	277	2391.11	307	473	512	887.33	512
35	hamming10-4	1024	104.68	31	1005.45	32	-	-	59.64	33
36	hamming6-2	64	0.0	32	0.02	32	32	32	0.18	32
37	hamming6-4	64	0.0	4	0.0	4	4	4	0.06	4
38	hamming8-2	256	0.68	96	10.61	111	128	128	11.17	128
39	hamming8-4	256	0.29	13	1.68	13	16	16	1.45	16
40	johnson16-2-4	120	0.05	8	0.29	8	8	8	0.2	8
41	johnson32-2-4	496	11.38	16	122.29	16	-	-	5.15	16
42	johnson8-2-4	28	0.0	4	0.0	4	4	4	0.01	4
43	johnson8-4-4	70	0.0	11	0.02	14	14	14	0.09	14
44	keller4	171	0.05	11	0.46	11	11	11	0.65	9
45	keller5	776	29.24	23	255.78	25	27	27	33.44	20
46	keller6	3361	18566.02	47	-	-	-	-	-	-
47	p_hat1000-1	1000	3.39	9	3.32	9	10	10	64.41	8
48	p_hat1000-2	1000	17.1	34	17.32	34	44	46	60.79	28
49	p_hat1000-3	1000	46.68	46	47.93	46	67	68	97.47	46

Table 1: Results and running time of TrustCLQ, nx.max_clique and ILS algorithms on the DIMACS graphs; t_{nx} denotes the running time of the nx.max_clique algorithm, $t_{0.1}$ and $t_{1.0}$ denotes the running time of the TrustCLQ algorithm for $SCALE = 0.1$ and $SCALE = 1.0$, respectively; ω_{best} is the best-known solution; ω_{ILS} denotes the solution obtained by the ILS algorithm; $\omega_{0.1}$ and $\omega_{1.0}$ denote the solutions obtained by the TrustCLQ algorithm for $SCALE = 0.1$ and $SCALE = 1.0$, respectively

Dataset	$\|V\|$	$t_{1.0}$	$\omega_{1.0}$	ω_{LB}
socfb-American75.mtx	6k	40.57	39	9
socfb-Amherst41.mtx	2k	5.98	21	10
socfb-Auburn71.mtx	18k	663.72	56	13
socfb-BC17.mtx	12k	200.13	35	12
socfb-BU10.mtx	20k	483.46	38	12
socfb-Baylor93.mtx	13k	319.47	54	11
socfb-Berkeley13.mtx	23k	832.31	42	15
socfb-Bingham82.mtx	10k	150.24	42	10
socfb-Bowdoin47.mtx	2k	4.94	22	9
socfb-Brandeis99.mtx	4k	16.74	23	9
socfb-Brown11.mtx	9k	140.75	32	10
socfb-Bucknell39.mtx	4k	18.38	29	9
socfb-CMU.mtx	7k	65.80	44	15
socfb-Cal65.mtx	11k	180.57	50	19
socfb-Caltech36.mtx	769	0.38	20	12
socfb-Colgate88.mtx	3k	15.82	33	12
socfb-Columbia2.mtx	12k	242.18	31	10
socfb-Cornell5.mtx	19k	653.97	40	13
socfb-Dartmouth6.mtx	8k	102.01	28	11
socfb-Duke14.mtx	10k	229.36	33	9
socfb-Emory27.mtx	7k	103.40	52	10
socfb-FSU53.mtx	28k	1346.47	54	24
socfb-Indiana.mtx	30k	1492.60	47	11
socfb-Indiana69.mtx	30k	1501.44	47	11
socfb-JMU79.mtx	14k	240.37	38	9
socfb-JohnsHopkins55.mtx	5k	31.62	44	9
socfb-Lehigh96.mtx	5k	34.60	37	10
socfb-MIT.mtx	6k	62.37	32	9
socfb-MSU24.mtx	32k	1565.69	46	11
socfb-MU78.mtx	15k	389.12	48	11
socfb-Maine59.mtx	9k	90.58	28	11
socfb-Maryland58.mtx	21k	652.87	53	9
socfb-Mich67.mtx	67k	8.76	27	14
socfb-Michigan23.mtx	30k	1539.24	44	11
socfb-Middlebury45.mtx	3k	11.00	24	13
socfb-Mississippi66.mtx	11k	262.77	47	10
socfb-NYU9.mtx	22k	706.92	37	13
socfb-Northeastern19.mtx	14k	252.21	34	11
socfb-Northwestern25.mtx	11k	219.91	40	21
socfb-NotreDame57.mtx	12k	291.54	25	12
socfb-Temple83.mtx	14k	332.03	35	10
socfb-Tennessee95.mtx	17k	832.38	57	9
socfb-Texas80.mtx	32k	2835.07	57	12
socfb-Texas84.mtx	36k	4410.97	50	9

Dataset	$\|V\|$	$t_{1.0}$	$\omega_{1.0}$	ω_{LB}
socfb-Trinity100.mtx	3k	15.25	36	11
socfb-Tufts18.mtx	7k	122.37	34	9
socfb-Tulane29.mtx	8k	172.76	38	16
socfb-UC33.mtx	17k	734.74	42	15
socfb-UC61.mtx	14k	538.35	46	18
socfb-UC64.mtx	7k	80.81	32	13
socfb-UCF52.mtx	15k	593.20	59	14
socfb-UCLA.mtx	20k	1282.03	51	9
socfb-UCLA26.mtx	20k	1284.71	51	9
socfb-UCSB37.mtx	15k	653.25	53	12
socfb-UCSC68.mtx	9k	160.68	30	12
socfb-UCSD34.mtx	15k	592.51	43	12
socfb-UChicago30.mtx	7k	102.23	33	9
socfb-GWU54.mtx	12k	190.39	42	9
socfb-Hamilton46.mtx	2k	5.56	25	15
socfb-Harvard1.mtx	15k	462.97	39	9
socfb-Howard90.mtx	4k	25.98	47	9
socfb-UConn.mtx	17k	404.61	49	9
socfb-UConn91.mtx	17k	405.48	49	9
socfb-UF.mtx	35k	2156.92	55	13
socfb-UF21.mtx	35k	2193.94	55	13
socfb-UGA50.mtx	24k	1153.77	51	12
socfb-UIllinois.mtx	31k	1663.48	56	9
socfb-UIllinois20.mtx	31k	1656.72	56	10
socfb-UMass92.mtx	17k	376.00	35	10
socfb-UNC28.mtx	18k	597.23	46	11
socfb-UPenn7.mtx	15k	447.09	41	10
socfb-USC35.mtx	17k	614.39	60	9
socfb-USF51.mtx	13k	214.36	43	10
socfb-USFCA72.mtx	3k	4.23	28	10
socfb-UVA16.mtx	17k	470.91	42	9
socfb-Vanderbilt48.mtx	8k	109.41	45	9
socfb-Vassar85.mtx	3k	9.47	22	10
socfb-Vermont70.mtx	7k	50.51	38	14
socfb-Villanova62.mtx	8k	87.20	36	9
socfb-Virginia63.mtx	21k	629.39	51	9
socfb-Wake73.mtx	5k	53.47	45	10
socfb-WashU32.mtx	8k	114.94	41	9
socfb-Wellesley22.mtx	3k	7.50	27	10
socfb-Wesleyan43.mtx	4k	14.84	28	19
socfb-William77.mtx	6k	66.31	38	10
socfb-Williams40.mtx	3k	9.44	24	14
socfb-Wisconsin87.mtx	24k	855.08	35	12
socfb-Yale4.mtx	9k	153.50	30	10

Table 2: Results of the TrustCLQ algorithm on Facebook networks. $t_{1.0}$ denotes the running time of the algorithm for $SCALE = 1.0$; ω_{LB} is the known lower bound on the size of the maximum clique; $\omega_{1.0}$ denotes the solution obtained by the TrustCLQ algorithm for $SCALE = 1.0$

Table 2 shows that the `TrustCLQ` algorithm allocates maximum cliques of a much larger sizes than the known lower bounds on the sizes of the maximum cliques. We also note that the running time of the algorithm is acceptable for solving problems of the approximate search for the maximum cliques on real social graphs.

4 Conclusion

In this paper, an heuristic algorithm for solving the maximum clique search problem is proposed. The algorithm uses both the well-known branch-and-bound method, as well as new ideas that were not previously mentioned by researchers. The algorithm uses an it-

erative approach for solving MCP and the results is practically independent of the order of vertices. In addition, it does not use the graph partition into independent sets.

The algorithm was tested on both artificial graphs and real networks. He showed not so high accuracy on DIMACS graphs, which are not real graphs, but are good for testing MCP algorithms. The algorithm showed good results when running on social networks, for which the obtained solution (i.e. the size of maximum clique) exceeded the known lower bound on the size of the maximum clique.

The algorithm was compared with two well-known algorithms. Each of them is good in individual cases. We can not say that the proposed algorithm is completely better in terms of the MCP solution. However, it works well on some graphs.

The future research may include the comparison of the proposed approach with the state-of-the-art methods described in [12] and [18].

Acknowledgements: The work has been supported by the Ministry of Science and Higher Education of Russian Federation (Project FSRR-2020-0006).

References:

[1] M.G.C. Resende and C.C. Ribeiro, *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer, 2016.

[2] Q. Wu and Jin-Kao Hao, A review on algorithms for maximum clique problems, *European Journal of Operational Research*, 242(3), 2015, pp. 693–709.

[3] E. Tomita and T. Seki, An efficient branch-and-bound algorithm for finding a maximum clique, In *Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science, DMTCS'03* 2003, pp. 278–289, Berlin, Heidelberg, 2003. Springer-Verlag.

[4] E. Tomita and T. Kameda, An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments, *Journal of Global Optimization* 37(1), 2007, pp. 95–111.

[5] E. Tomita, Y. Sutani, T. Higashi and M. Wakatsuki, A simple and faster branch-and-bound algorithm for finding a maximum clique with computational experiments, *IEICE Transactions on Information and Systems* E96.D(6), 2013, pp. 1286–1298.

[6] J. Konc and D. Janezic, An improved branch and bound algorithm for the maximum clique problem, *MATCH - Communications in Mathematical and in Computer Chemistry* 58, 2007, pp. 569–590.

[7] M. Depolli, J. Konc, K. Rozman, R. Trobec and D. Janezic. Exact parallel maximum clique algorithm for general and protein graphs, *Journal of chemical information and modeling* 53(9), 2013, pp. 2217–28.

[8] P. San Segundo, D. Rodriguez-Losada and A. Jimenez, An exact bit-parallel algorithm for the maximum clique problem, *Computers & Operations Research* 38, 2011, pp. 571–581.

[9] B. Pattabiraman, Md. Mostofa Ali Patwary, A. H. Gebremedhin, W.-K. Liao and A. Choudhary, Fast algorithms for the maximum clique problem on massive sparse graphs, Eds. A. Bonato, M. Mitzenmacher and P. Prałat, *Algorithms and Models for the Web Graph*, 2013, pp. 156–169, Cham, 2013. Springer International Publishing.

[10] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin and Md. M. A. Patwary, A fast parallel maximum clique algorithm for large sparse graphs and temporal strong components, *CoRR*, abs/1302.6256, 2013.

[11] P. Daniluk, G. Firlik and B. Lesyng, Implementation of a maximum clique search procedure on cuda, *Journal of Heuristics* 25(2), 2019, pp. 247–271.

[12] C.-M. Li, H. Jiang and F. Manyà, On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem, *Computers & Operations Research* 84, 2017, pp. 1–15.

[13] R. Battiti and M. Protasi, Reactive local search for the maximum clique problem1, *Algorithmica* 29(4), 2001, pp. 610–637.

[14] W. Pullan and H. H. Hoos, Dynamic local search for the maximum clique problem, *J. Artif. Int. Res.* 25(1), 2006, pp. 159–185.

[15] W. Pullan, F. Mascia and M. Brunato, Cooperating local search for the maximum clique problem, *Journal of Heuristics* 17(2), 2011, pp. 181–199.

[16] S. Busygin, A new trust region technique for the maximum weight clique problem, *Discrete Applied Mathematics* 154(15), 2006, pp. 2080–2096.

[17] Q. Zhang, J. Sun and E. Tsang, An evolutionary algorithm with guided mutation for the maximum clique problem, *IEEE Transactions on Evolutionary Computation* 9(2), 2005, pp. 192–200.

[18] Y. Wang, S. Cai and M. Yin, Two Efficient Local Search Algorithms for Maximum Weight Clique Problem, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)* 2016, pp. 805–811.

[19] Y. Wang, S. Cai, J. Chen and M. Yin, SCCWalk: An efficient local search algorithm and its improvements for maximum weight clique problem, *Artificial Intelligence* 280, 2020, pp. 103230.

[20] P. S. Segundo, S. Coniglio, F. Furini and I. Ljubic, A new branch-and-bound algorithm for the maximum edge-weighted clique problem, *European Journal of Operational Research* 278(1), 2019, pp. 76–90.

[21] T. Yu and M. Liu, A memory efficient maximal clique enumeration method for sparse graphs with a parallel implementation, *Parallel Computing* 87(1), 2019, pp. 46–59.

[22] L. Chang, Efficient maximum clique computation and enumeration over large sparse graphs, *The VLDB Journal* 2020, https://doi.org/10.1007/s00778-020-00602-z

[23] P. S. Segundo, F. Furini and J. Artieda, A new branch-and-bound algorithm for the Maximum Weighted Clique Problem, *Computers & Operations Research* 110, 2019, pp. 18–33.

[24] Y. Chu, B. Liu, S. Cai, C. Luo and H. You, An efficient local search algorithm for solving maximum edge weight clique problem in large graphs, *Journal of Combinatorial Optimization* 2020, https://doi.org/10.1007/s10878-020-00529-9

[25] B. Nogueira and R.G.S. Pinheiro, A GPU based local search algorithm for the unweighted and weighted maximum s-plex problems, *Ann. Oper. Res.* 284, 2020, pp. 367–400.

[26] E. Sevinc and T. Dokeroglu, A novel parallel local search algorithm for the maximum vertex weight clique problem in large graphs, *Soft Comput.* 24, 2020, pp. 3551–3567.

[27] H. Jiang, C.-M. Li, Y. Liu and F. Manya, A Two-Stage MaxSAT Reasoning Approach for the Maximum Weight Clique Problem, *Proc. of Thirty-Second AAAI Conference on Artificial Intelligence* 2018, pp. 1338–1346.

[28] Y. Wang, S. Cai and M. Yin, New heuristic approaches for maximum balanced biclique problem, *Information Sciences* 432, 2018, pp. 362–375.

[29] D. V. Andrade, M. G. C. Resende and R. F. Werneck, Fast local search for the maximum independent set problem, *Journal of Heuristics* 18(4), 2012, pp. 525–547.

[30] L. A. Sanchis, Generating hard and diverse test sets for np-hard graph problems, *Discrete Applied Mathematics* 58(1), 1995, pp. 35 – 66.

[31] M. Batsyn, B. Goldengorin, Eu. Maslov and P. Pardalos, Improvements to mcs algorithm for the maximum clique problem, *Journal of Combinatorial Optimization* 27, 2014, pp. 397–416.

[32] R. A. Rossi and N. K. Ahmed, The network data repository with interactive graph analytics and visualization, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015, pp. 4292–4293.