ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 2, February 2015

Survey of Algorithms on Maximum Clique Problem

Krishna Kumar Singh[1], Dr. Ajeet Kumar Pandey[2]

Lecturer, RGUKT- Nuzvid, AP, India[1]

AECOM Hyderabad, AP, India[2]

Abstract: The maximum clique problem (MCP) is to determine a sub graph of maximum cardinality. A clique is a sub graph in which all pairs of vertices are mutually adjacent. Based on existing surveys, the main goal of this paper is to provide a simplified version and comprehensive review on Maximum clique problem. This review intends to encourage and motivate new researchers in this area. Though capturing the complete literature in this regard is beyond scope of the paper, but it is tried to capture most of the representative papers from similar approaches.

Keywords: Maximum Clique problem, Exact Algorithms, Approximation Algorithms, Heuristic Approach, Local search.

1. INTRODUCTION

A complete sub graph of G is one whose vertices are pair wise adjacent. Finding a clique of a fixed size k is a well-known NP-complete problem known as k-clique [1]. The corresponding optimization problem i.e. finding the maximum complete sub graph of G is known as the Maximum Clique Problem (MCP). Maximum cliques size of a graph is also called clique number of the graph and is represented by ω(G). Maximum clique problem is equivalent to the independent set problem as well as to the minimum vertex cover problem. There are many application areas on MCP; Bioinformatics [2; 3], Social Networks [4], coding theory, fault diagnosis, geometry, computer vision and pattern recognition etc. as surveyed in [5], etc. Since it is NP-hard problem, so it is difficult to obtain a polynomial time algorithm to find exact solution, but with proper understanding of previous work some better algorithm may be devised to achieve significant performance, and using heuristic based greedy approach, near optimal solution can be found efficiently.

The rest of the sections are presented as follows, Section 3 discusses about the review of related works. The paper is concluded in Section 4.

2. NOTATIONS

Given a graph G(V, E), where V is the set of vertices and E is the set of edges, d(v) is degree of vertex $v \epsilon V$, and N(v) is a set of neighbors of vertex v. U is candidate set to be searched to improve clique size at each recursive search step. C is a clique under improvement and updated locally within the iteration. C* is the clique and is updated globally at the end of each iteration, i.e. if $|C|>|C^*|$ then $C^*=C$. N(C) is set of vertices which are connected to all vertices in C. $N_1(C)$ is set of vertices which are connected to all vertices in C, but one (except any one of them).

3. REVIEW OF RELATED WORKS

Algorithms for NP-hard problems typically fall into one of three categories: exact algorithms, approximation algorithms, and Heuristic approach. The following sections describe each.

3.1. EXACT ALGORITHMS

A brute force algorithm to test whether a graph *G* contains a *k*-vertex clique, and to find any such clique, is to examine each sub-graph with at least *k* vertices and check to see whether it forms a clique. This algorithm takes time O $(n^k k^2)$, since there are $O(n^k)$ sub graphs to check, each of which has $O(k^2)$ edges whose presence in *G* needs to be checked.

The progress in computing technologies in 1960's motivated the development of many new enumerative algorithms. **Bron and Kerbosch** [6] proposed a backtracking method that requires only polynomial storage space and excludes all the possibility of computing the same clique twice, as shown in the Algorithm-1 below. Tomita et al. [7] developed a modification of this approach that has the time complexity of O $(3^{n/3})$.

Algorithm 1: Classic Bron-Kerbosch algorithm

```
Bron-Kerbosch (C, U, X)
  1 if U and X are both empty then
  2     report C as a maximal clique
//choose the pivot vertex u in U ∪ X as the vertex with the highest number of neighbors in P
   3 for each vertex v in U do
   4      Bron-Kerbosch (C ∪ v, U ∩ N(v), X ∩ N(v))
   5     U ← U \ v
   6     X ← X ∪ v
```

Initially C and X is set to Φ, and U contains all the vertexes the graph. At each recursive step, C is the temporary result, U is the set of the possible candidates set and X is excluded set. N(v) indicates the neighbors of the vertex v. The algorithm works as follow: Pick a vertex v from U to expand. Add v to C and remove its non-neighbors from U and X. Then pick another vertex from the new U and repeat the process. Continue until U is empty. Once U is empty, if X is empty then report the content of C as a new maximal clique (if it's not then C contains a subset of an already found clique). Now backtrack to the last vertex picked and restore U, C and X as they were before the choice, remove the vertex from U and add it to X, then expand the next vertex. If there is no more vertexes in U then backtrack to the superior level.

David Eppstain demonstrated [8] is a modified version of the Bron-Kerbosch (including the comment mentioned in the algorithm 1) that visits the graph using a degeneracy ordering, and can be bound in complexity to O ($d * n * 3^{d/3}$). Given d, degeneracy of the graph, an ordering in which each vertex has d or less neighbors which can be found out in linear time.

The above-mentioned algorithms were designed for finding all maximal cliques in a graph, but solving the maximum clique problem requires finding only one maximum clique, which are described in subsequent sections.

Tarjan and Trojanowski [9] proposed a recursive algorithm for the maximum independent set problem with the time complexity of O ($2^{n/3}$). Finding Maximum independent set in a graph G, is equivalent to finding maximum clique in its (G) complement graph (G'). An improved result to obtain the time complexity of O ($2^{0.288n}$) is presented in [10]. Robson [11] further improved the best known worst-case complexity to O ($2^{n/4}$). Exact algorithms are guaranteed to return optimal results. However all known exact algorithms for MAXCLIQUE are relatively slow as they take exponential time in the worst case. They mainly are based on the backtracking and Branch& Bound framework.

A well-known exact algorithm (denoted by EA; enumerative algorithm) is developed by **Carraghan and Pardalos** in [12] which is shown in Algorithm-2. Despite its simplicity, this algorithm constitutes an important step for exact solving of the MCP and provides the basis for many later improved exact clique algorithms. The functioning of this algorithm is discussed below in detail.

Algorithm 2: Branch &Bound algorithm

```
Function clique (U; C)
1:  if |U| =0 then
2:      if |C| >| C*| then
3:          |C*|:=|C|
4:          New record; save it.
5:      end if
6:      return
7:  end if
8:  while U ≠Ø do
9:      if |C| + |U| < |C*| then
10:         return
11:     end if
12:   i:=min{j | v_j∈U}
13:   U:=U-{v}
14:   clique(U ∩ N(vi); C ∪i)
15:  end while
16:  return
function old
17:  C*:=Φ
18:  clique (V; Φ)
19:  return
```

Vertex set V is first ordered and one by one vertex is explored. The vertices of G is ordered into a list L = (v_1, v_2..., v_n) where v_n is the vertex of minimum degree in G, v_{n-1} is the vertex of minimum degree in G-{v_n}, and v_{n-2} is the vertex of minimum degree in G-{v_n,v_{n-1}}, and so on. Once the maximum clique induced on a particular vertex (v_i) is found, it (v_i) is removed from the ordered list. In other word, every time N(v) is found in right side from itself of the ordered list. Each vertex of U is connected to all the vertices of C, i.e. any vertex v of U can be added to C to obtain a larger clique C'= C ∪ {v}. The pruning is done when the set U (current candidate set) becomes so small that even if all vertices in U would be added to the C (current local clique size), the size of that clique would not exceed that of the largest clique found previously.

Osterg'ard [13] proposed a better heuristic algorithm (it is named here REA, i.e. reverse enumerative algorithm) with reverse ordering as in algorithm-2, and improved the upper bound of the EA algorithm described above. It uses an additional memory to store the clique size induced on each of the vertex and latter it is used while pruning the branch. The algorithm remembers the maximum clique found for each vertex previously into a special array *b*. So *b*[*i*] is the maximum clique for the *i*-th vertex while searching backward. This number is used later as: if we search for a clique of size greater than |C*|, then the search on v_i is pruned if v_i is going to be the (j + 1)-th vertex in C and $j + b[i] \leq |C^*|$.

To estimate the upper bound of the maximum clique, graph coloring techniques are also applied to the subgraph induced by the candidate set U. This is based on a general fact that if a graph can be colored with k colors, then the maximum clique in this graph must be smaller or equal to k. Using color classes instead of b[i] (mentioned above) improves the upper bound and consequently reduces the size of the search tree. In addition, vertex coloring is also a NP-hard problem and may be expensive so, a greedy method may be used for coloring the vertex set U during the search process. The following (next two) algorithms of exact solution, unlike EA and REA, are based on color based pruning.

Deniss Kumlander [14] proposes a better heuristic based vertex coloring and backtracking for MCP. The algorithm works like REA, mentioned above, but the pruning condition is different. Initially vertices are sorted by color classes obtained by a heuristic vertex coloring algorithm, i.e. $V = \{C_n, C_{n-1}, ..., , C_1\}$, where C_i is a set of vertices with I, i.e. *i*-th color class. First of all cliques that could be built on vertices in C_1 are explored. Then on vertices of C_1 and C_2, i.e. of the first and second color classes, and so forth. In other word; at the *i*-th step all cliques can that contain vertices of $\{C_i, C_{i-1}, ..., , C_1\}$, are explored. The algorithm remembers the maximum clique found for each for each color class into a special array *b*. So *b*[*i*] is the maximum clique for a subgraph formed by $\{C_i, C_{i-1}, ..., C_1\}$ vertices while searching backward. This is used later for pruning the branch; if max clique size found so far is $|C^*|$, then if *vi* is going to be (j + 1)-th vertex in C and it belongs to the *k*-th colour class and $j + b[k] \leq |C^*|$, then the branch is pruned.

An improved version of greedy coloring based algorithm is proposed as **MCS** [15], which uses a recoloring strategy using greedy approximate coloring procedure and outperform its predecessors in [16; 17]. Vertices of U are sorted in an ascending order with respect to the color number. Then, at each search step of the algorithm, MCS selects a vertex v ∈ U in reverse order (the last vertex in the reordered set U belongs to the highest color class) and the color number associated with each vertex becomes an upper bound for the maximum clique in the remaining subgraph in the current branch to be searched. The basic idea of pruning in MCS is that if a vertex $v \in U$ with color number $Kv< (|C^*| - |C|)$, then the vertex v needs not to be searched from. The number of vertices to be searched can be further reduced, as to move vertex v with color $>(|C^*| - |C|)$ to other color class less than $(|C^*| - |C|)$ in number. This is how number of vertices to be searched in the candidate set U is reduced.

A constraint programming (CP) based B&B algorithm is proposed in [18, 19], it provides a heuristic to filter the vertices which is not going to lead to a better clique size. The approach that can be viewed as an adaptation and a generalization of the Bron & Kerbosh's [6] ideas for enumerating the maximal cliques of a graph. The upper bound of clique size in every branching step is computed based on a matching algorithm rather than a coloring algorithm. For each vertex v in U, the upper bound of clique size roughly corresponds to the number of vertices in N (v) minus matching number in the subgraph (its complement graph) induced by N(v). If this upper bound plus the current clique size is smaller than the maximum clique obtained in previous all branching steps, then v can be removed from U. Concluding the section of exact algorithm for MCP, the Table-1 summarizes the reviewed algorithms

3.2. APPROXIMATION ALGORITHMS

Approximation algorithms guarantee a result within a certain range of the optimal solution and typically run faster than exact algorithms. Much theoretical work has been done to determine the extent to which MAXCLIQUE is approximable. Hastad's [20] proved that there is no polynomial time approximation for MAXCLIQUE within a factor of n^{1-k} for any $k > 0$ given a graph with n vertices, unless NP = P. A greedy algorithm that builds a maximal independent set by recursively adding a minimum degree vertex and removing its neighbors has an approximation ratio of $(\Delta+2)/3$ [21]. But the current best-known polynomial-time approximation algorithm achieves only an approximation guarantee of $O(n.(\log\log n)^2 /(\log n)^3)$ [22].

3.3. HEURISTIC APPROACHES TO THE MCP

In addition to exact and approximation algorithms, significant progresses on heuristic algorithms for the MCP also have been proposed in the recent years. In the following sections, the most representative heuristics of the problem are reviewed.

3.3.1. Greedy Algorithms

A simple greedy algorithm for MAXCLIQUE is illustrated in Figure 1. Greedy algorithms are frequently used in practice for their simplicity of implementation and better efficiency. In greedy heuristics, decisions on vertex to be added in or moved out are usually based on some static information associated with the vertices in the candidate set like their degrees. Several improvements to the static greedy heuristics have been proposed in the literature. For instance QUALEX-MS [23] and DAGS [24], are described below.

Table-1: Comparative view of major exact algorithm of MCP

Algorithm Name	**Author Name**	**Year**	**Approach/ Methodology**	**Remarks on Performance**
Classic Bron-Kerbosch algorithm	Bron, Coen; Kerbosch, Joep	1973	backtracking method, excludes all the possibility of computing the same clique twice	$O(3^{n/3})$, enumerates all maximal cliques.
MIS/MCP	R. E. Tarjan and A. E. Trojanowski	1977	The algorithm uses a recursive, or backtracking scheme and concept of connected component and dominance	$O(2^{n/3})$, it is much faster than enumeration of all independent set.
EA	R. Carraghan, P.M. Pardalos	1990	The basic B&B approach	A landmark B&B algorithm, provides basis for many later B&B algorithms.
REA	P.R.J. Östergård	2002	Based on B & B, Uses reverse ordering of vertices as in EA	Performance is better than EA
CPR	Jean-Charles Regin	2003	Constraint programming, Using B&B and filtering of vertices to tighten the candidate set U	Faster than another B&B algorithm using filtering algorithms on most DIMACS[1] instances
VColor-BT-u	Deniss Kumlander	2006	Based on Color Classes and Backtracking	Outperform EA and REA
MCS	E. Tomita, et. al.	2010	B&B based on subgraphs coloring	An improved version of subgraph coloring algorithm, performs better than it's all predecessors on many DIMACS instances

[1] The DIMACS benchmark set is created in 1990's for the second DIMACS challenge on Clique [34], consisting of 80 graphs on satisfiability, Graph Coloring, and MCP. ftp://dimacs.rutgers.edu/pub/challenge/graph/

Table-2: Comparative view of major heuristic algorithm of MCP

Algorithm Name	Author Name	Year	Approach/ Methodology	Remarks on Performance
RLS	Battiti & Protasi	2001	Reactive Tabu local search, past-sensitive scheme to determine the amount of diversification	A landmark MCP algorithm, reports better results than its predecessors
KLS	Katayama, Hamamoto, & Narihisa,	2004	Iterated local search	Achieves a good performance on the MANN instances from DIMACS but a bad performance on the keller and brock instance
DAGS	Grosso, A., Locatelli, M., & Croce, F. D.	2004	Greedy algorithm including plateau search.	One of the best performing algorithms, shows highly competitive results compared with a number of state-of-the-art algorithms before DLS
QUALEX-MS	S. Busygin	2006	greedy algorithm, vertex weights are derived from a nonlinear programming formulation	Better Performance on Brock instances of DIMACS.
DLS	Pullan W, Hoos H	2006	Dynamic local search, a simplified DAGS including perturbation strategies.	One of the best performing algorithms, shows highly competitive results compared with a number of state-of-the-art algorithms before DLS
PLS	Pullan W	2006	Phased based local search, a robust form of DLS	performances comparable to or better than DLS on the DIMACS benchmarks, except Keller6
CLS	W. Pullan, F. Mascia, M. Brunato	2011	Cooperating local search, advance than PLS and paralleled algorithm	Shows excellent performances on both DIMACS and BOSHLIB benchmarks. One of the current best performing MCP algorithms
ELS	S. Balaji	2013	Edge based local search for vertex cover and equivalently for MCP	Performance on both DIMACS and BOSHLIB benchmarks is equivalent or better than PLS.
TALS	K K Singh, K K Naveena, G Likhitaa	2014	Target aware local search, Incorporating prohibition time for diversification.	Outperform DLS excluding some instances of P-hat, gen-400, keller-6 of DIMACS benchmarks.

QUALEX-MS [26] is a deterministic iterated greedy construction algorithm that uses vertex weights derived from a nonlinear programming formulation of MAXCLIQUE. In QUALEX-MS, each vertex is assigned a weight which represents its importance towards inclusion in the improving clique. Vertex weights are calculated on the basis of the coordinates of stationary points of nonlinear programs derived from the Motzkin-Straus nonlinear formulation of the MCP.

The **Deep Adaptive Greedy Search (DAGS)** algorithm [24] uses an iterated greedy construction procedure. Starting from basic greedy heuristics, modifications and improvements are combined in a two-phase heuristic procedure. In the first phase an improved greedy procedure is applied starting from each node of the graph; Based on number of occurrence of vertices a reduced subset of nodes is selected, and in second phase an adaptive greedy algorithm is applied to find more promising cliques around such nodes. A generic Add move procedure is shown in Figure-1, and swap move is shown in Figure-2. Swap move is used to fine Plateau (Clique with same size by swapping a vertex from C, current clique). DAGS adaptively adjusts the vertex weights used for vertex selection by a restart mechanism in order to guide the search towards less explored areas. Computational results show that DAGS is superior to QUALEX-MS for most of the tested DIMACS instances.

3.3.2. Local Search

Local search is a sophisticated way of using greedy approach. However, greedy algorithms can easily fall into the local optima due to their short-sighted nature. Several improvements to the greedy heuristics [2, 3, 4] have been proposed in the literature.

Although most algorithms have been empirically evaluated on benchmark instances from the Second DIMACS Challenge but, somewhat unsurprisingly, there is no single best algorithm. Nevertheless, there are few heuristic MAX-CLIQUE algorithms, described in the subsequent sections that achieved state-of-the-art performance.

```
Procedure Greedy_Add($v \in V$)
1. Set $C = C \cup \{v\}$;
2. set $N(C)=N(v)$
2. while $N(C) \neq \emptyset$ do
3.     Select $i \in N(C)$
       Such that $|N(i) \cap N(C)|$ is maximum;
4.     Set $C = C \cup \{i\}$;
5. end while;
6. return C; //C is clique induced on v
```

Figure 1: A simple greedy procedure called in Clique improvement phase

```
Procedure Greedy_swap_Move (v∈V)

1.  Select i ∈N_1(C)
        Such that |N (i) ∩ N(C)| is maximum or at random
2.  Set C = C ∪{i};
3.  Remove/drop vertex v from C, which is not adjacent to i
4.  Store the removed vertex v in tabu list,
            // vertices in tabu list are prohibited till a
        specified no. of iteration.
4.  Update N(C) & N_1(C), excluding the vertices in tabu list.
```

Figure 2: A simple greedy procedure called for Plateau search

A Reactive Local Search (RLS) [25] is a landmark algorithm to the MCP which is based on local search complemented by a feedback (past-sensitive) scheme to determine the amount of diversification. RLS applies add moves whenever this is possible and selects a vertex with the highest number of adjacent vertices in current Candidate set to add to the clique C, similar as shown in Figure-1. If no allowed addition exists, RLS searches for an allowed vertex to drop from C such that its removal leads to the larger set of vertex additions to C. If no allowed moves are available, a random vertex is picked from C and is dropped. As soon as a vertex is moved, it is put into the tabu list and remains prohibited for the next T iterations. The prohibition period T is related to the amount of desired diversification, and is determined by feedback information from the search history.

The k-opt algorithm (KLS) [26] is based on a conceptually simple variable depth search procedure that uses elementary search steps in which a vertex is added to or removed from the current clique. It is also called Iterated KLS (IKLS for short). It consists of three components: Local Search at which KLS is used, a Kick called LEC-Kick that escapes from local optima, and Restart that occasionally diversifies the search by moving to other points in the search space. Drop moves are considered only when no add or swap move exists. There is some evidence that it performs better than RLS on many instances from the DIMACS benchmark sets.
The **Dynamic Local Search (DLS-MC)** [27] scheme is actually a slight simplification of the Deep Adaptive Greedy Search of [24]. The algorithm works as follows: After picking an initial random vertex and setting it to the current clique, all vertices penalties are initialized to zero. Then, the search alternates between an iterative improvement phase, during which suitable vertices are repeatedly added to the current clique C, and a plateau search phase, as shown in Figure-2, in which repeatedly one vertex of C is swapped with a vertex currently not contained in C. In the case of expand, the selection is made from the set N(C); it is called improving neighbor set of C, as shown in Figure-1. In plateau Search, on the other hand, the vertex to be added to C is selected from $N_1(C)$, which comprises the vertices that are connected to all vertices in C except for one vertex, say v′, which is subsequently removed from C. vertex penalties are used throughout the entire search, a “forgetting” mechanism decreasing the penalties is added. When no add and swap moves are possible, some perturbation strategies are applied to the clique C. The perturbation mechanism chooses a random vertex or the last selected vertex and is added in C, and then removes all vertices from C that are not connected to v and then the process of iterative improvement phase and plateau search phase is repeated alternatively. The authors report a very good performance on the DIMACS instances. The algorithm is realized through efficient supporting data structures leads to smaller overall CPU times.
The family of stochastic local search algorithms, dynamic local search (DLS) [27], phased local search (PLS) [28] and cooperating local search (CLS) [29], share similar strategies of clique expansion, plateau search, and search stagnation.

The phased Based local search (PLS) [28], to cope with graphs of different structures, combines three sub-algorithms which use different vertex selection rules: random selection, random selection among those with the highest vertex degree, and random selection within those with the lowest vertex penalty. For each of the sub problem the procedure of DLS-MC is adopted. The performance results for PLS with respect to DLS-MC shows that, excluding keller6, PLS is either comparable or more efficient than DLS-MC for all DIMACS instances.

The Cooperating Local search (CLS) [29] further improves over PLS by incorporating four low level heuristics which are effective for different instance types. CLS is a parallel maximum clique hyper-heuristic that incorporates four low level heuristics, namely: Greedy Search (GREEDY) which uses random selection within vertex degree, is biased towards higher degree vertices and performs limited plateau search; Level Search (LEVEL) which uses random selection within vertex degree, is biased towards higher degree vertices and performs extensive plateau search; Focus Search (FOCUS) which obtains an average vertex degree for the clique as close as possible to this focus vertex degree, and, Penalty Search (PENALTY), which further enhance the performance of CLS, relevant information is passed between low level heuristics in order to guide the search to particular areas of the search domain. CLS shows excellent performances on both DIMACS and BOSHLIB benchmarks.

The edge based local search (ELS) algorithm [35] is a two phased local search method for MCP. The ELS works as follows: In first phase, the algorithm greedily constructs a vertex cover and the second phase a pruning technique is applied to make the vertex cover more optimal. A parameter ’support’ of vertices defined in the ELS greatly reduces the number of random selections of vertices and also the number of iterations and running times. While constructing optimal vertex cover, the vertex with more support is chosen greedily first. For each v ∈ V in the graph G, support of a vertex is defined by

$$s(v)=d(v)+\sum_{u\in N(v)}d(u)$$

The quantity $\sum_{u\in N(v)}d(u)$ is the sum of the degree of vertices which are adjacent to v. The computational results on BHOSLIB and DIMACS benchmark graphs indicate that ELS is capable of achieving state-of-the-art-performance for the maximum clique with reasonable average running times

The Target Aware Local Search for Maximum Clique Problem (TALS) [30] is a multi restart based and improved local search techniques. The diversification in order to avoid the local optima is achieved by incrementing the prohibition time for selected vertex in the current clique so that in next restart, the vertex is delayed from being selected for a certain number of iteration. It starts with a clique C with single vertex and iteration is initialized by zero. Iteratively a vertex from $N_1(C)$ is selected with maximum degree and its prohibition time is lesser than the current iteration value. Subsequently the iteration is incremented by one, and prohibition time of the selected vertex is incremented by some adjusted integral value, which later decreases as iteration is increases. If there is no more vertices in $N_1(C)$ and if target clique is not achieved, then the process restart with a new initial vertex. TALS outperforms other algorithm like RLS, DLS-MC for most of the DIMACS graphs.

In the past few years, many powerful variations of the basic local search procedure have been developed, many of which are inspired by natural occurring phenomena. Examples of such algorithms are simulated annealing [32], and genetic algorithm [33]. Surprisingly all these techniques have been applied to the maximum clique problem. As concluding remark of the section, the reviewed algorithms are summarized in the Table-2.

4. CONCLUSION

Based on the review of various algorithms, summarization of similar algorithms is tabulated; additionally we take a privilege to comment on future enhancement of the algorithms. In the view of exact solution of MCP, further enhancement may be done by applying better coloring algorithm to estimate upper bound. In the view of local search technique, we can see that hardly a algorithm dominates on all other algorithm, it is because of diverse structure of graphs. One possible way to overcome this deficiency may be to incorporate multiple search operators within a single algorithm and incorporating dynamic capability to decide the most permissible operators to be triggered during the search process.

REFERENCES

[1]. Karp, R.M: In: Miller, R.E., Thatcher, J.W. "Reducibility among Combinatorial Problems", pp. 85–103. Plenum, New York, 1972.
[2]. Bahadur D.K.C., Akutsu, T., Tomita, E., Seki, Fujiyama, "Point matching under non-uniform distortions and protein side chain packing based on an efficient maximum clique algorithm", Genome Informatics, 13, pp. 143–152, 2002
[3]. Etsuji Tomita, Tatsuya Akutsu and Tsutomu Matsunaga," Efficient Algorithms for Finding Maximum and Maximal Cliques: Effective Tools for Bioinformatics", ISBN 978-953-307-475-7, 2011.
[4]. B. Balasundaram, S. Butenko, I.V. Hicks," Clique relaxations in social network analysis: The maximum k-plex problem", Operations Research, 59(1): pp. 133–142, 2011.
[5]. Pardalos, P.M., Bomze,I.M., Budinich,M. and Pelillo,M. " The Maximum Clique Problem. in Handbook of Combinatorial Optimization, Supplement", Vol. A, Kluwer Academic Publishers: 1-74, 1999
[6]. Bron, Coen; Kerbosch, Joep, Algorithm 457, "finding all cliques of an undirected graph", Commun. ACM (ACM) 16 (9): pp. 575–577, 1973.
[7]. Tomita, Etsuji; Tanaka, Akira; Takahashi, Haruhisa, "The worst-case time complexity for generating all maximal cliques and computational experiments", Theoretical Computer Science 363(1): pp. 28–42, 2006.
[8]. Eppstein, David; Strash, Darren, "Listing all maximal cliques in large sparse real-world graphs", 10th International Symposium on Experimental Algorithms, 2011.
[9]. R. E. Tarjan and A. E. Trojanowski. "Finding a maximum independent set", SIAM Journal of Computing, 6: pp. 537–546, 1977.
[10]. F. V. Fomin, F. Grandoni, and D. Kratsch. "Measure and conquer: domination – a case study". In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, Proc. ICALP'05, volume 3580 of Lecture Notes in Computer Science, pp. 191–203. SpringerVerlag, 2005.
[11]. J. M. Robson. "Finding a maximum independent set in time $O(2^{n/4})$", Technical Report 1251-01, LaBRI, Université de Bordeaux I, 2001.
[12]. R. Carraghan, P.M. Pardalos, "An exact algorithm for the maximum clique problem", Oper. Research.Letters. Vol. 9 pp 375–382, 1990.
[13]. Östergård, P.R.J., "A fast algorithm for the maximum clique problem". Volume 120, Issues 1–3, pp. 197–207, 2002.
[14]. Deniss Kumlander, "A simple and efficient algorithm for the maximum clique finding reusing a Heuristic vertex colouring". IADIS international journal on computer science and information systems, Vol. 1, No. 2 , pp. 32-49, ISSN: 1646-3692, 2006.
[15]. E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, M. Wakatsuki, "A simple and faster branch-and bound algorithm for finding a maximum clique". Lecture Notes in Computer Science, volume 5942, pp. 191–203, 2010.
[16] E. Tomita, T. Seki, "An efficient branch-and-bound algorithm for finding a maximum clique". Lecture Notes in Computer Science, volume 2731, pp, 278–289, 2003.
[17] E. Tomita, T. Kameda, "An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments". Journal of Global Optimization, 37(1): 95–111, 2007
[18]. Kluwer, M. Milano, "Using Constraint Programming to solve the Maximum Clique Problem", editor Constraints and Integer Programming, 2003.
[19]. Jean-Charles Regin, "Solving the Maximum Clique Problem with Constraint Programming", Proceedings CPAIOR, 2003
[20]. J. H'astad, "Clique is hard to approximate within n^{1-e} ". Acta Mathematica, 182(1):105–142, 1999.
[21]. M. M. Halld´orsson and J. Radhakrishnan, "Greed is good: Approximating independent sets in sparse and bounded-degree graphs". Algorithmica, 18:145–163, 1997.
[22]. U. Feige, "Approximating maximum clique by removing subgraphs". SIAM Journal on Discrete Mathematics, 18(2):219–225, 2004.
[23]. S. Busygin, "A new trust region technique for the maximum weight clique problem", Discrete Applied Mathematics, 154(15): 2080–2096, 2006.
[24]. Grosso, A., Locatelli, M., & Croce, F. D, "Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem". Journal of Heuristics, 10, pp. 135–152, 2004
[25]. Battiti, R., & Protasi, M. "Reactive local search for the maximum clique problem", Algorithmica, 29, 610–637, 2001.
[26]. Katayama, K., Hamamoto, A., & Narihisa, H. (2004). "Solving the maximum clique problem by k-opt local search". In Proceedings-ACM Symposium on Applied computing, pp. 1021–1025, 2004.
[27]. Pullan W, Hoos H., " Dynamic local search for the maximum clique problem". Journal of Artificial Intelligence Research 25, pp. 159-185, 2006.
[28]. Pullan, "Phased local search for the maximum clique problem", J Comb Optimum 12:303–323, 2006.
[29]. W. Pullan, F. Mascia, M. Brunato, "Cooperating local search for the maximum clique problem". Journal of Heuristics, 17(2): 181–199, 2011.
[30]. K K Singh, K K Naveena, G. Likhitaa "Target Aware Local Search for Maximum Clique Problem", IFRSA's International Journal Of Computing, Vol. 4 Issue. 3, 2014.
[31]. S. Balaji, " A New Effective Local Search Heuristic for the Maximum clique problem ", World Academy of Science, Engineering and Technology, International Journal of Mathematical, Computational, Physical and Quantum Engineering Vol:7 No:5, 2013
[32]. Xiutang Geng, Jin Xu, Jianhua Xiao, Linqiang Pan, "A simple simulated annealing algorithm for the maximum clique problem", Information Sciences 177, 5064–5071, 2007.
[33]. A.S. Murthy, G. Parthasarathy and V.U.K. Sastry, "Clique finding-A genetic approach", Proc. 1st IEEE Conf. Evolutionary Computing.: 18-21, 1994.
[34]. D.S. Johnson and M.A. Trick (eds.), "Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge", DIMACS Vol. 26, American Mathematical Society, 1996 (see also http://dimacs.rutgers.edu/Volumes/Vol26.html).