

Modified Approach for Solving Maximum Clique Problem.

A Dissertation submitted

By Jaspreet Singh
Reg No - 41100104
Section - K2112

To

Department of Computer Science and Engineering

In partial fulfillment of the requirement for the

Award of the Degree of

Master of Technology in Computer Science and Engineering

Under the guidance of

Mr. Kiran Kumar Kaki
(Assistant Professor)

May-2014

DECLARATION

I hereby declare that the dissertation entitled — “**Modified Approach for Solving Maximum Clique Problem**” submitted for the M.Tech degree is entirely my original work and all references and ideas have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date: 01/12/2013

Investigator: Jaspreet Singh

Reg no: 41100104

ABSTRACT

Graph Theory is the branch of Mathematics and Theoretical Computer Science which has great contributions to Computations and many applications in the real world. Finding Maximum Clique from a graph is a process of extracting connected components of graph. A connected component is a sub-graph having path or reachability of each node to every other node in it. This dissertation work contains exploration of maximum number of connected components of an input graph. For extracting maximum number of cliques from a graph an algorithm MODIFIEDCLIQUE is proposed. MATLAB framework has been chosen for simulating this work, because MATLAB provides graph viewing functions producing real like graph structure for visualization unlike other frameworks producing graphs just like array of nodes. CPU elapsed times of my algorithm are quite good. However it performs moderately, not much well as like other peer algorithms. This algorithm is tested on various standard graphs like Hamming20, Keller6 and Brock20 etc. For Hamming20 it shows CPU elapsed time as 0.81121 milliseconds.

Keywords: MCP, BK, ACMCP, MODIFIEDCLIQUE, Hamming20, Keller6, Brock20.

CERTIFICATE

This is to certify that **Jaspreet Singh (41100104)** has completed M.Tech dissertation titled **Modified Approach for Solving Maximum Clique Problem** under my guidance and supervision. To the best of my knowledge, the present work is the result of his original investigation and study. No part of the dissertation has ever been submitted for any other degree or diploma.

The dissertation is fit for the submission and partial fulfilment of the conditions for the award of M.Tech. computer science and engineering.

Date: _____

Signature of Advisor: _____

Mr. Kiran Kumar Kaki
(Assistant Professor)

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to Mr. Kiran Kumar Kaki (Dissertation Mentor) and Mrs. Harjeet Kaur for their valuable knowledge and expertise. It is only with their guidance that I could take up the initiative of such a good topic of thesis and complete it on time. I am also very thankful to Lovely Professional University for giving me an opportunity to propose and implement my work.

I am gratified for the successful completion of my thesis implementation. I would also like to convey thanks to all my friends who gave their full support and encouraged me for this thesis work.

Jaspreet Singh

41100104

TABLE OF CONTENTS

S No.	Topic	Page No.
Chapter 1	Introduction	1-11
	1.1 Maximum Clique Problem	1
	1.2 Genetic Algorithms	2
	1.3 Outline of basic Genetic Algorithm	5
	1.4 Maximum Clique Problem is NP Hard	6
	1.5 Applications of Maximum Clique Problems	6
	1.6 Other Techniques of finding Maximum Clique Problems.	10
Chapter 2	Literature Review	12-20
Chapter 3	Present Work	21-23
	3.1 Significance	21
	3.2 Objective	21
	3.3 Methodology	22
	3.4 Sources of Data	23
	3.5 Research Design	23
Chapter 4	Dissertation Monthly Progress	24-25
	4.1 Gantt Chart	24
Chapter 5	Results and Discussions	26-40
	5.1 Read time and Run time of Standard graphs of DIMACS	26
	5.2 Results of HGAMC algorithm	27
	5.3 Comparison of MODIFIEDCLIQUE results with other algorithms	28
	5.3.1 Comparison Graph of MODIFIEDCLIQUE results with other algorithms	29
	5.4 Comparison of MODIFIEDCLIQUE cputimes with other algorithms	30
	5.5 Comparing Performance of various Standard graphs	31
	5.6 Generation of Adjacency Matrix in MATLAB	32

	5.7	Viewing graph in Bio-Informatics	33
	5.8	Realization of Input Graph	34
	5.9	Designing interface for Modified Clique finding Approach	35
	5.10	Implementation of MODIFIEDCLIQUE method	36
	5.11	Result containing Maximum Cliques	37
	5.12	Result showing each node's containability and No. of components	38
	5.13	Input Vectors on Command Window	39
	5.14	Command window showing S and C	40
Chapter 6		Conclusion and Future Scope	41
Chapter 7		References	42-43
Chapter 8		Appendix	44-48
	8.1	Glossary of terms	44
	8.2	Abbreviation	47

TABLE OF FIGURES

Figure No.	Description	Page No.
1.1	An example of Clique.	2
1.2	Roulette Wheel Selection.	3
1.3	Mutation	4
1.4	Crossover	4
1.5	Genetic Algorithm Cycle	5
3.1	Research Design	23
5.1	Comparison of MODIFIEDCLIQUE number of cliques.	29
5.2	Performance graph of various standard graphs	31
5.3	Adjacency Matrix for input graph	32
5.4	Input graph	33
5.5	Plot of Adjacency Sparse Matrix	34
5.6	Fig file of Interface Design	35
5.7	Interface ModifiedClique	36
5.8	Maximum Cliques	37
5.9	Listing Cliques with their number	38
5.10	Vectors on Command Window	39
5.11	Results in Command Window	40

TABLE OF TABLES

Table No.	Description	Page No.
4.1	Gantt Chart	24-25
5.1	Read time and Run time of Standard graphs	26
5.2	Results of HGAMC algorithm	27
5.3	Comparison of HGAMC results with other algorithms	28
5.4	Comparison of MODIFIEDCLIQUE cputimes with other algorithms	30

1.1 Maximum Clique Problem

Assume that the finite undirected simple graph $G = (V, E)$ is given, where V is the set of nodes, $V \in N$, E is the set of edges. The arbitrary full graph is called a clique which is a sub-graph of large graph. This sub-graph is fully connected with each node accessible from every other node. The clique, which does not contain other cliques, is called a maximal Clique. The largest maximal clique is called a maximum clique. To extract all maximal cliques from the graph G . Many algorithms have been described to solve this problem. The best solution now a days is a procedure where the complexity is linear to the number of maximal cliques [1,2]. The theory and algorithms described in this paper can solve the problem. We assume that the graph G is presented in the form of an adjacency matrix $X : N \times N$, the main diagonal of which has zeros. Given an undirected graph $G = (V, E)$, a clique S is a subset of V such that for any two elements $u, v \in S$, $(u, v) \in E$. Using the notation E_S to represent the subset of edges which have both endpoints in clique S , the induced graph $G_S = (S, E_S)$ is complete. Finding Maximum clique in a graph is an NP-hard problem, called the maximum clique problem (MCP). Cliques are intimately related to vertex covers and independent sets. Given a graph G , and defining E^* to be the complement of E , S is a maximum independent set in the complementary graph $G^* = (V, E^*)$ if and only if S is a maximum clique in G . That means for a complementary graph, maximum clique remains same as in original graph. Hence for an undirected graph maximum clique is represented by full matrix but for directed graph it is upper triangular or lower triangular matrix. It follows that $V - S$ is a minimum vertex cover in G^* .

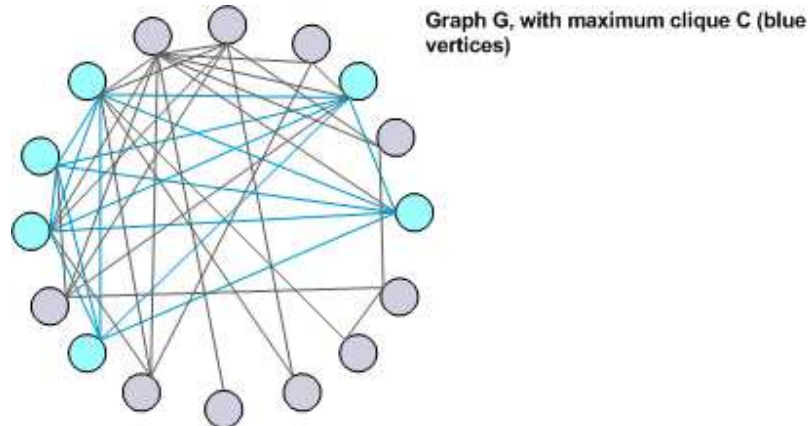


Fig. 1.1 An Example of Clique.

In other words a **clique** in an undirected graph $G = (V, E)$ is a subset of the vertex set $C \subseteq V$, such that for every two vertices in C , there is an edge connecting the two vertices. This is equivalent to saying that the sub graph induced by C is complete. A maximal clique is a clique that cannot be extended by including one more adjacent vertex to it, means and a clique which does not exist exclusively within the vertex set of a larger clique. A maximum clique is a clique of the largest possible size in a given graph. The clique number $\omega(G)$ of a graph G is defined the number of vertices in a maximum clique in G . The intersection number of G denoted by $T(G)$ is also termed as the smallest number of cliques that altogether cover all edges of G . The opposite of a clique is observed as an independent set, in the sense that every clique which corresponds to an independent set in the complement graph. The cliques cover problem concerns with finding as few cliques as possible that include every vertex in the graph. A related concept is a bi-clique, a complete bipartite sub graph. The bipartite dimension of a graph is the minimum number of bi-cliques needed to cover all the edges of the graph.

1.2 GENETIC ALGORITHMS

Genetic algorithms are the computation model closest to natural evolution. Their success at searching complex non-linear spaces and general robustness has led to their use in a number of practical problems such as scheduling, financial modeling and optimization. The inventor of genetic algorithms, John Holland, took his inspiration for them from nature. Genetic algorithms contain a population of individuals, each of which has a known fitness. The population is evolved through successive generations; the individuals in each new generation are bred from the

fitter individuals of the previous generation. Unlike Natural Evolution which is continuously indefinite, we have to decide when to stop our GA. As with the breeding of domestic animals, we choose the individuals to breed from to drive the population's evolution in the direction we want it to go. As with domestic animals, it may take many generations to produce individuals with the required characteristics. Inside a computer an individual's fitness is usually calculated directly from its DNA and so only the DNA need to be represented. Usually genetic algorithms represent DNA by a fixed length vector. Where a genetic algorithm is being used for optimization, each individual is a point in the search space and is evaluated by the fitness function to yield a number indicating how much right that point is. If any point is good enough, the genetic algorithm stops and the solution is simply that point. If not then a new population, containing the next generation is bred.

The breeding of a new generation is inspired by nature; new vectors are bred from the fitter vectors in the current generation, using either asexual or sexual reproduction. In asexual reproduction, the parent vector is simply copied.

Chromosomes are selected from the population to be parents to crossover. The problem here is that how to select these chromosomes. According to Darwin's evolution theory *survival of the fittest*, the best ones should survive and create new offspring. There are many methods that how to select the best chromosomes, for example Roulette wheel selection, Boltzmann selection, Tournament selection, rank selection, steady state selection and some others. Among these methods Roulette Wheel Selection is widely used for selection process in genetic algorithms. Parents are selected according to their fitness. The better the chromosomes are, the more chances they have to be selected. Imagine a roulette wheel where are placed all chromosomes in the population, every chromosome has its place more according to its fitness function.

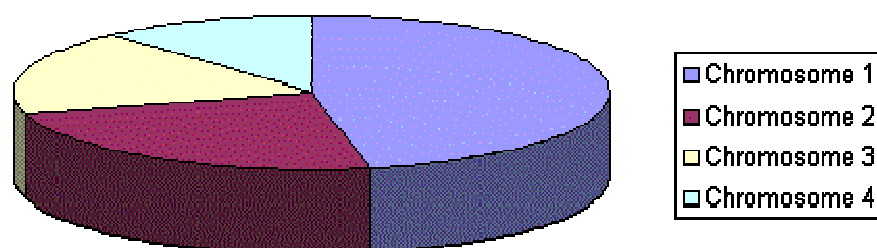


Fig. 1.2 Roulette Wheel Selection

Then a marble is thrown there to select the chromosome. Chromosome with bigger fitness will be selected more times. Figure 3 shows a child vector being created by mutating a single gene where each gene is represented by a single bit. There are more chances for chromosomes with bigger fitness to be selected when it roulette wheel is rotated under probability distribution techniques. With sexual reproduction, two of the fitter vectors are chosen and the new vector is created by sequentially copying sequences alternately from each parent. Typically only two or three sequences are used, and the point(s) where the copying crosses over to the other parent is chosen at random. This is known as crossover. Crossing over these bit patterns are simply representing changing features of successors from their predecessors. Figure 4 shows a child being formed firstly by copying four genes from the left-hand parent then the three remaining genes are copied from the right-hand parent.

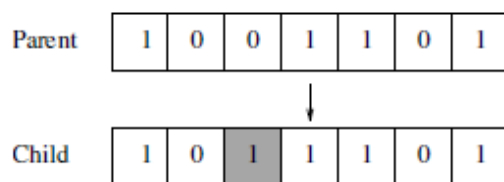


Fig. 1.3 Mutation

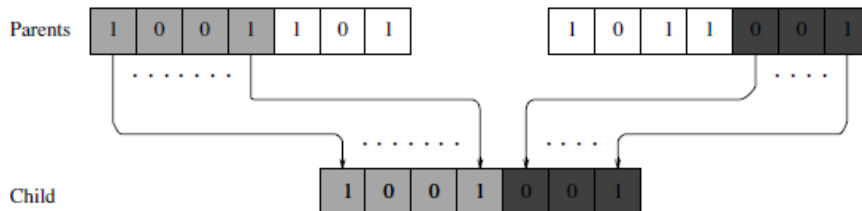


Fig. 1.4 Crossover

Holland in his paper "Genetic Algorithms and the Optimal Allocation of Trials" [Hol73] shows, via his schemata theorem, that in certain circumstances genetic algorithms make good use of information from the search so far to guide the choice of new points to search. Figure 5 shows the genetic algorithm cycle. The schemata theorem requires the vector representation and fitness function be designed so that the required solution can be composed of short fragments of vectors which, if present in a vector, give it a relatively high fitness regardless of the contents of the rest

of the vector. These are known as building blocks. They can be thought of as collections of genes which work well together.

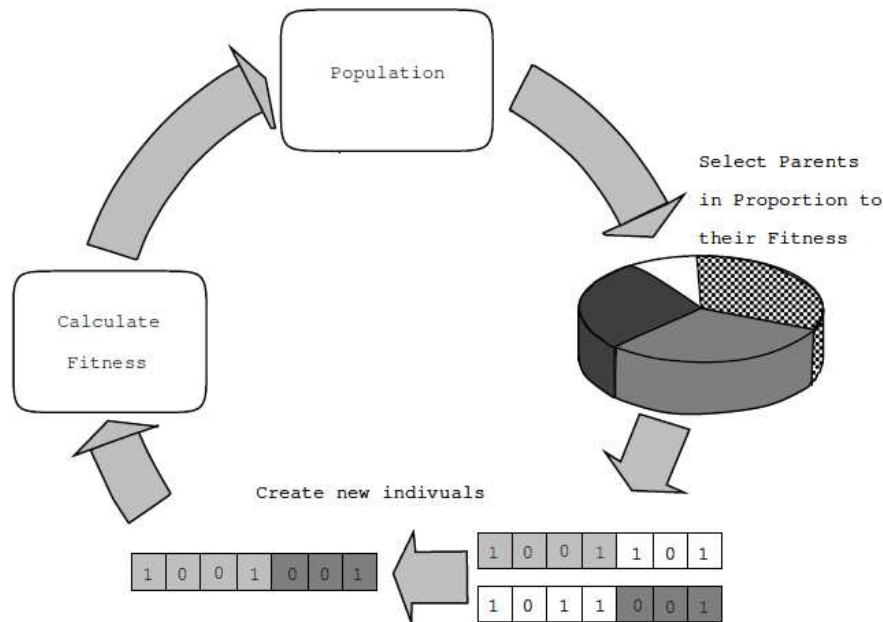


Fig. 1.5 The Genetic Algorithm Cycle.

1.3 OUTLINE OF THE BASIC GENETIC ALGORITHM

1. **[Start]** Generate random population of n chromosomes (suitable solutions for the problem)
2. **[Fitness]** Evaluate the fitness $f(x)$ of each chromosome x in the population
3. **[New population]** Create a new population by repeating following steps until the new population is complete
 - a. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
 - b. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
 - c. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
 - d. **[Accepting]** Place new offspring in a new population
4. **[Replace]** Use new generated population for a further run of algorithm

5. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
6. **[Loop]** Go to step 2

1.4 MAXIMUM CLIQUE PROBLEM IS NP HARD

Examples of difficult problems, which cannot be solved in "traditional" way, are NP problems. There are many tasks for which we know fast (polynomial) algorithms. There are also some problems that are not possible to be solved algorithmically. For some problems was proved that they are not solvable in polynomial time. But there are many important tasks, for which it is very difficult to find a solution, but once we have it, it is easy to check the solution. This fact led to **NP-complete** problems. NP stands for nondeterministic polynomial and it means that it is possible to "guess" the solution (by some nondeterministic algorithm) and then check it, both in polynomial time. If we had a machine that can guess, we would be able to find a solution in some reasonable time. Studying of NP-complete problems is for simplicity restricted to the problems, where the answer can be yes or no. Because there are tasks with complicated outputs, a class of problems called **NP-hard** problems has been introduced. This class is not as limited as class of NP-complete problems. For NP-problems is characteristic that some simple algorithm to find a solution is obvious at a first sight - just trying all possible solutions. But this algorithm is very slow (usually $O(2^n)$) and even for a bit bigger instances of the problems it is not usable at all. Today nobody knows if some faster exact algorithm exists. Proving or disproving these remains as a big task for new researchers. Today many people think, that such an algorithm does not exist and so they are looking for some alternative methods – example of these methods are genetic algorithms. Examples of the NP problems are Maximum Clique Problem, Travelling Salesman Problem or Knapsack Problem.

1.5 APPLICATIONS OF MAXIMUM CLIQUE PROBLEM

The MODIFIEDCLIQUE Problem has many real world applications. It is encountered in many different fields in which either the underlying problem can be formulated as the MODIFIEDCLIQUE problem or finding the maximum clique is a precondition of solving the

problem. Based on those applications, a collection of much diversified test graphs for the MODIFIEDCLIQUE problem has been created for evaluating the performance of algorithms for the MODIFIEDCLIQUE problem. They are available at

<ftp://dimacs.rutgers.edu/pub/challenge/graph/>

And consist of graphs derived from different problems such as coding theory, fault diagnosis and printed circuit board testing.

1.5.1 Coding theory

A common problem in coding theory is to find a binary code as large as possible that can correct a certain number of errors for a given binary word. A binary code is a set of binary vectors. The Hamming distance between two binary vectors is defined as the number of positions in which the two vectors have different values. A maximum clique of $H(n,d)$ represents the maximum number of binary vectors of size n with Hamming distance greater than or equal to d . Therefore, if we find the maximum clique C in $H(n,d)$, any binary code consisting of vectors represented by the vertices in C is able to correct $(d-1)/2$ errors.

1.5.2 Fault diagnosis

Fault diagnosis plays a very important role in studying the reliability of large multiprocessor systems. The goal is to identify all faulty processors (units) in the system. In the model designed by Berman and Pelc [1], the system is represented by an undirected graph $G = (V, E)$ whose vertices are processors and where edges are communication links.

1.5.3 Printed circuit board testing

A printed circuit board tester involves placing probes onto a board. A probe can determine if a portion of a board is working correctly. Since probes have a particular size, not every component can be checked in one pass. The problem of maximizing the number of components checked in one pass can be formulated as a clique problem: each node connects a component and an edge represents two nodes that are not too close to be checked simultaneously. A clique in this graph is then a set of components that can be checked in one pass.

1.5.4 Web Communities

Consider a directed graph $G = (V, A)$ (called *web network*) whose vertices and arcs correspond to web pages and their links, respectively. Kumar et al. [16] regarded *directed bipartite cliques* $(S1, S2)$ (i.e., $S1 \times S2 \in A$) of G as communities of web pages, i.e., the web pages in $S2$ may have similar topics and web pages in $S1$ may have interests in these topics, and considered generating directed bipartite cliques of G . They first construct a graph G^* with about 5,000,000 arcs by removing unnecessary vertices and arcs from G , and then enumerate all directed bipartite cliques in the reduced graph G^* . They show that directed bipartite cliques usually contain similar topics by checking them by human hands. However, since G^* contains a great number of bipartite cliques, they could enumerate only those containing at most 10 vertices. In this setting, it is natural to regard maximal directed bipartite cliques as good representatives of communities. From a directed graph $G = (V, A)$, let us construct a bipartite (undirected) graph $G^1 = (V \cup V^1, E)$ such that $V^1 = \{v^1 \mid v \in V\}$ is a copy of V and $(v, u^1) \in E$ if and only if $(v, u) \in A$. Then there exists a one-to-one correspondence between directed bipartite cliques in G and bipartite cliques in G^1 . Hence, our algorithms are applicable to generate all maximal directed bipartite cliques in G^* .

1.5.5 Basic Algorithms

In this section, the algorithms of Tsukiyama et al. [18] and Johnson et al. [17] are considered. It is viewed that their algorithms as the enumeration algorithms based on reverse search, where *reverse search* was introduced by Avis and Fukuda [19] to solve enumeration problems efficiently. Note that our presentation of their algorithms is quite different from theirs [18], which may be of independent interest. Let K_0 denote the maximal clique that is the lexicographically largest among all maximal cliques. For a maximal clique K ($\neq K_0$), we define a *parent* $P(K)$ of K by $C(K \leq i-1)$ such that i is the maximum index satisfying $C(K \leq i-1) \neq K$. Such an index i is called the *parent index*, denoted by $i(K)$. Note that they are well-defined, since $K \neq C(K \leq 0)$ holds by $K \neq K_0$. Since $P(K)$ is lexicographically larger than K , this parent-child binary relation on maximal cliques is acyclic, and creates an in-tree rooted by K_0 .

Lemma 1. *The parent-child relation constructs an in-tree rooted by K_0 .*

It is called in-tree the *enumeration tree* for maximal cliques of a graph G . Both algorithms [17] traverse this enumeration tree. In order to traverse enumeration tree, we have to compute a parent and children of a given maximal clique efficiently. It is not difficult to see that a parent $P(K)$ is computable from a maximal clique K in linear time. However, it is not so trivial to compute from K its children. For a maximal clique K and an index i , we define $K[i] = C(K_{<i} \cap \Gamma(v_i)) \cup \{v_i\}$).

Lemma 2. *Let K and K' be maximal cliques in G . Then K' is a child of K if and only*

if $K' = K[i]$ holds for some i such that

- (a) $v_i \notin K$.
- (b) $i > i(K)$.
- (c) $K[i]_{<i-1} = K_{<i} \cap \Gamma(v_i)$.
- (d) $K_{<i} = C(K_{<i} \cap \Gamma(v_i))_{<i}$.

Moreover, if an index i satisfies (a) ~ (d), then i is the parent index of $K[i]$.

Since $C(K)$ can be computed from a clique K in $O(m)$ time, by Lemma 2, we can compute all children of a given maximal clique in $O(nm)$ time. Therefore, we can traverse the enumeration tree efficiently. The algorithm of Tsukiyama et al. traverses the enumeration tree in a depth-first manner. Their algorithm starts with a root K_0 , and find its children recursively. It is not difficult to see that the algorithm requires $O(nm)$ time delay and $O(n + m)$ space. The algorithm of Johnson et al. enumerates all maximal cliques in the lexicographically decreasing order. Their algorithm initializes a queue Q as $Q = \{K_0\}$, iteratively extracts the lexicographically largest element K from Q and inserts into Q all the children which are lexicographically smaller than K . The time complexity of their algorithm is same as the algorithm of Tsukiyama et al., however, it needs $O(nN + m)$ space, where N denotes the number of all maximal cliques.

1.6 OTHER TECHNIQUES OF FINDING MAXIMUM CLIQUE PROBLEMS

1.6.1 Simulated Annealing

Simulated annealing is a randomized neighborhood search algorithm inspired by the physical annealing process, where a solid is first heated up in a heat bath until it melts, and then cooled down until it solidifies into a low-energy state. It was first introduced by Kirkpatrick, Gelatt and Vecchi in 1983 [7]. This heuristic technique considers the solutions of a combinatorial optimization problem corresponding to the states of the physical system and the cost of a solution is equivalent to the energy of the state. A simulated annealing algorithm basically works as follows. First, a tentative solution in the state space is generated usually at random. Then the next state is produced from the current one. The new state is evaluated by a cost function f . If it improves, the new state is accepted. If not, the new state is accepted with probability $e^{-\Delta f/\tau}$, where Δf is the difference of the cost function between the new state and the current state, and τ is a parameter usually called the temperature in analogy with physical annealing, which is varied during the optimization process [8]. The simulated annealing heuristic has been ranked among one of the best heuristics for the MODIFIED CLIQUE problem at the 1993 DIMACS challenges [6].

1.6.2 Neural Networks

An Artificial Neural Network (ANN) is a parallel system inspired by the densely interconnected, parallel structure of the mammalian brain information- processing system [5]. Some mathematical models of the biology nervous systems show that the temporal evolution is controlled by a quadratic Lyapunov function (also called energy function), which is iteratively minimized as the process evolves. This feature can be applied to many combinatorial optimization problems. More than ten algorithms have been proposed for solving the MAX- CLIQUE problem using neural networks. They encode the problem in different models, but most of them are based on the Hopfield model [5] and its variations. The problem is solved via several discrete (deterministic and stochastic) and continuous energy-descent dynamics. In general, algorithms based on neural networks can find significantly larger cliques than other simpler heuristics but the running time is slightly longer. On the other hand, comparing to those more

sophisticated heuristics, they obtained significantly smaller cliques on average but were considerably faster [8].

1.6.3 Tabu Search

Tabu search is a modified local search algorithm, in which a prohibition-based strategy is employed to avoid cycles in the search trajectories and to explore new regions in the search space [8]. A tabu list is used to store historical information on the search path to prevent the algorithm from going back to recently visited solutions. Tabu solutions are accepted if they satisfy some aspiration level condition. Several tabu search algorithms for the MODIFIED CLIQUE problem have been developed in the past ten years. They basically have the same structures but change the definition of the search space, the ways that tabu lists are used and the aspiration mechanism. In Battiti and Protasi's algorithm [3], a reactive local search method is used so that the size of the tabu list can be automatically determined. Also, an explicit restart procedure influenced by memory is activated to introduce diversification. The worst case complexity per iteration of this algorithm is $O(\max(|V|, |E|))$ where V is the vertex set and E is the edge set of the graph. The running time of the algorithm is better than those presented at the Second DIMACS Implementation Challenge [6]. There are also many simple heuristics that have been used to solve the MAX-CLIQUE problem such as the sequential greedy heuristics and local search heuristics. They usually have better running times than those advanced heuristics algorithms discussed above, but the quality of the results is worse on the average.

CHAPTER 2

LITRATURE REVIEW

This observation was first mathematically formulated by John Holland in 1975 in his paper, "Adaptation in Natural and Artificial Systems" [5]. Usually the algorithm breeds a predetermined number of generations; each generation is populated with a predetermined number of fixed length binary strings. These binary strings are then translated (decoded) into a format that represents suitable parameters either for some controller, or as an output.

The product resulting from evolution (whether natural or simulated) is not simply discovered by a random search through the problem state space, but by a directed search from random positions in that space. In fact, according to Goldberg, the simulated evolution of a solution through genetic algorithms is, in some cases, more efficient and robust than the random search, enumerative or calculus based techniques. The main reasons given by Goldberg are the probability of a multi-modal problem state space in non-linear problems, and that random or enumerative searches are exhaustive if the dimensions of the state space are too great [4].

In the earliest work, BronKerbosch algorithm was designed by Dutch scientists Joep Kerbosch and Coenraad Bron, who published a description of their algorithm in 1973. The basic form of the algorithm was inefficient in the case of graphs with many non-maximal cliques. It makes a recursive call for every clique whether it is maximal or not. To save time and allow the algorithm to backtrack more quickly in branches of the search that contain no maximal cliques, Bron and Kerbosch introduced a variant of the algorithm involving a "pivot vertex". Any maximal clique must include either that pivot vertex or one of its non-neighbors, for otherwise the clique could be augmented by adding pivot to it. Therefore, only pivot and its non-neighbors need to be tested as the choices for any other vertex that is added to clique in each recursive call to the algorithm. Although other algorithms for solving the clique problem have running times that are in theory, better on inputs that have few maximal independent sets but the BronKerbosch algorithm and subsequent improvements to it are frequently reported as being more efficient in practice than the alternatives. It is well-known and widely used in application areas of graph algorithms such as computational graph theory. In computer science, the BronKerbosch algorithm is an algorithm

for finding maximal cliques in an undirected graph. That is, it lists all subsets of vertices with the two properties that each pair of vertices in one of the listed subsets is connected by an edge, and no listed subset can have any additional vertices added to it while preserving its complete connectivity [11]. For sparse graphs, tighter bounds are possible. In particular the vertex-ordering version of the Bron–Kerbosch algorithm can be made to run in time $O(dn3^{d/3})$, where d is the degeneracy of the graph, a measure of its sparseness. There exist d -degenerate graphs for which the total number of maximal cliques is $(n - d)3^{d/3}$, so this bound is close to tight.

David R. Wood, “An Algorithm for finding maximum clique in a graph”, 1997 Elsevier Science, introduced a branch-and-bound algorithm for the maximum clique problem which applies existing clique finding and vertex coloring heuristics to determine lower and upper bounds for the size of a maximum clique[9]. Mr. Wood presented his algorithm which uses Fractional Coloring as heuristics to find upper and lower bounds. Since a color class can contain at most one vertex of a clique, in a fractional coloring the sum of the weights of those color classes intersecting a clique Q is at least $|Q|$. Therefore, the total weight of a fractional coloring of a graph G is an upper bound for $\omega(G)$. Vertex colorings provide much tighter upper bounds. A vertex coloring (or k -coloring) of a graph $G = (V, E)$ is a partition of V into independent sets (C_1, C_2, \dots, C_k) called color classes. Each C_i contains those vertices assigned the i^{th} color. A k -coloring of G must color each vertex of a clique differently, so k is an upper bound for $\omega(G)$. This algorithm include non-uniform random graphs with relatively large clique sizes, and graphs which have arisen in coding theory, the Steiner Triple Problem, tiling of hyper-cubes, vertex cover problems and fault diagnosis.

Patric R. J. Ostergard, “A Fast Algorithm for the maximum clique problem”, 2002 Elsevier Science, given a branch-and-bound algorithm for the maximum clique problem—which is computationally equivalent to the maximum independent (stable) set problem—is presented with the vertex order taken from a coloring of the vertices and with a new pruning strategy. The algorithm performs successfully for many instances when applied to random graphs and DIMACS benchmark graphs [10]. In the maximum clique problem, one desires to find one maximum clique of an arbitrary undirected graph. This problem is computationally equivalent to some other important graph problems, for example, the maximum independent (or stable) set problem and the minimum vertex cover problem. Since these are NP-hard problems [14], no

polynomial time algorithms are expected to be found. Nevertheless, as these problems have several important practical applications, it is of great interest to try to develop fast, exact algorithms for small instances. Another direction of research, which has recently been fairly popular, is that of using stochastic methods to find as large cliques as possible, without proving optimality; see the survey of Pardalos and Xue [15], which also contains an extensive bibliography on the maximum clique problem.

Xinshun Xu, Jun Ma, Jingsheng Lei, “Ant Colony Optimization for the Maximum Clique Problem” IEEE-ICNC 2007 introduced an evolutionary approach in which main task is to search for maximum cost path in a graph. Artificial Ants walk through graph and looking for high quality paths. Better results are found as emergent result of global cooperation among ants in colony.

Algorithm 1. Old algorithm.

function clique(U, size)

Step 1: **if** |U|=0 **then**

Step 2: **if** size > max **then**

Step 3: max := size

Step 4: New record; save it.

Step 5: **end if**

Step 6: **return**

Step 7: **end if**

Step 8: **while** U != \emptyset **do**

Step 9: **if** size + |U| < max **then**

Step 10: **return**

Step 11: **end if**

Step 12: $i := \min\{j \mid v_j \in U\}$

Step 13: $U := U \setminus \{v_i\}$

Step 14: $\text{clique}(U \cap N(v_i), \text{size} + 1)$

Step 15: **end while**

Step 16: **return**

function old

Step 17: $\text{max} := 0$

Step 18: $\text{clique}(V, 0)$

Step 19: **return**

The set of vertices adjacent to a vertex v is denoted by $N(v)$ and the number of vertices in the graph is n . The variable max , which is global, gives the size of a maximum clique when the algorithm terminates. The performance of the algorithm depends on the ordering the vertices, v_1, v_2, \dots, v_n . We will return to heuristic for ordering later. Each vertex taken in step 12 should be saved to be able to extract the whole clique whenever step 4 is reached. Without the pruning strategy in step 9 (in implementing the algorithm, the steps 8–11 can be combined into a for statement), this algorithm would go through every single clique of the graph. The pruning strategy is to backtrack when the set U becomes so small that even if all vertices left could be added to get a clique, the size of that clique would not exceed that of the largest clique encountered so far in the search. Moreover, if we explicitly search for a clique of a given size, we can modify the algorithm and use this information for pruning from the beginning of the search. Some speed-up can be obtained if the test in step 1 is changed so that the recursion is stopped whenever very few vertices are left (often 0 or 1) and corresponding calculations are carried out on a case-by-case basis. Although this algorithm is very simple, it is currently the best known algorithm for sparse graphs. Most attempts to improve on this straightforward algorithm are based on methods for calculating upper bounds (other than from the size of the set U in Algorithm 1) during the search. Almost without exceptions, such bounds are obtained from vertex-colorings. In vertex-coloring, adjacent vertices must be assigned different colors. If a

graph, or an induced subgraph, can be colored with, say, s colors, then the graph, or subgraph, cannot contain a clique of size $s + 1$. In implementing strategies based on calculating upper bounds, a trade-off has to be made; coloring can lead to a considerable reduction of the number of nodes in the search tree but is also very time-consuming. [10]

Algorithm 2. New algorithm.

function clique(U , size)

Step 1: **if** $|U| = 0$ **then**

Step 2: **if** size > max **then**

Step 3: max := size

Step 4: New record; save it.

Step 5: found := true

Step 6: **end if**

Step 7: **return**

Step 8: **end if**

Step 9: **while** $U \neq \emptyset$ **do**

Step 10: **if** size + $|U| \leq$ max **then**

Step 11: **return**

Step 12: **end if**

Step 13: $i := \min\{j \mid v_j \in U\}$

Step 14: **if** size + $c[i] \leq$ max **then**

Step 15: **return**

Step 16: **end if**

```

Step 17:    U := U \ {vi}
Step 18:    clique(U ∩ N(vi); size + 1)
Step 19:    if found=true then
Step 20:        return
Step 21:    end if
Step 22: end while
Step 23: return
function new
Step 24: max:=0
Step 25:    for i:=n down to 1 do
Step 26:        found:=false
Step 27:        clique(Si ∩ N(vi), 1)
Step 28:        c[i]:=max
Step 29:    end for
Step 30:    return

```

The function $c(i)$ gives the largest clique in S_i . Obviously, for any $1 \leq i \leq n-1$, we have that $c(i) = c(i+1)$ or $c(i) = c(i+1) + 1$. Moreover, we have $c(i) = c(i+1) + 1$ iff there is a clique in S_i of size $c(i+1) + 1$ that includes the vertex v_i . So, starting from $c(n) = 1$, we search for such cliques. If a clique is found, $c(i) = c(i+1) + 1$, otherwise $c(i) = c(i+1)$. The size of a maximum clique is given by $c(1)$. Old values of the function $c(i)$ enables the new pruning strategy (in step 14). Namely, if we search for a clique of size greater than s , then we can prune the search if we consider v_i to become the $(j + 1)$ -th vertex and $j + c(i) \leq s$.

Li Lu, Yunhong Gu, Robert Grossman, “dMaximalCliques: A Distributed Algorithm for Enumerating All Maximal Cliques and Maximal Clique Distribution” IEEE- ICDMW-2010 presents a distributed algorithm which can obtain clique information from million-node graphs. It has used distribution of size of maximal cliques in a graph as a new measure for measuring structural properties of a graph. The main reason for this algorithm to make the problem solvable in a distributed fashion is the locality property of the clique that is the information for finding all maximal cliques including v only depends on the neighbors of v . Experimental experience shows that this distributed method for finding maximum cliques takes more time for listing maximal cliques from the one which is generated by using the BFS method compared with that by using the random method. The basic concept is that, usually a vertex and its neighbors share similar characteristics, such as the degree. When they are grouped in the same surroundings, if the vertex has a large degree, so do its neighbors, the average size of sub-graph containing that vertex is large and further results in the unbalanced load when processing it. In contrast, the random method does not have this problem. The average size of this kind of sub-graph is approximately the same.

R.Rama, Suresh Badarla and Kamala krithivasan, “Clique-detection algorithm using clique-self-assembly”, IEEE BIC-TA.2011, proposed a brute force algorithm where a large graph is being decomposed and these decomposed parts are cliques [12]. Self-assembly is the process in which relatively simple components brought into contact with each other experience local interactions guided by basic rules, and combine to form increasingly complex structures. This process is quite useful in making determination of large molecules of protein as per the behavior of small molecules of which they are made. There is no externally guiding force or direction, just the summation of the undirected local interactions. The progress in science has experienced many of the naturally occurring self-assembling systems. These molecular structures are to be shown by using graph data structures and many graphs searching algorithm yielding fruitful results these days. Modeling molecules as labeled graphs has a long tradition and is a prerequisite for most modern Cheminformatic methods. The representation of molecules by graphs has two major advantages: Graphs are a very intuitive molecular representation close to our elementary chemical understanding, and they form a solid theoretical basis for computer –aided processing. Furthermore, graphs enable a database retrieval via graph isomorphism technique, i.e., comparing molecules becomes equivalent to comparing labeled graphs. The structural formula is

a graph-like representation of molecules commonly used to formulate and exchange chemical knowledge.

Hakan Yildiz, Christopher Kruegel, “Detecting Social Cliques for Automated Privacy Control in Online Social Networks”, Fourth International Workshop on Security and Social Networking, Lugano (19 March 2012) proposed a privacy control approach that addresses this problem by automatically detecting social cliques among the friends of a user. To find cliques, given a small number of friends (seed), uses the structure of the social graph to generate an approximate clique that contains this seed. [22] The cliques found by the algorithm can be transformed directly into friend lists, making sure that a piece of sensitive data is exposed only to the members of a particular clique. A social clique is a group of people having significant social interaction with each other due to a particular cause e.g., families, classmates, colleagues. A piece of data is often of concern only to a particular social clique. One can attempt to identify this clique by examining the data itself. Most data shared on social networking sites contains some information that identifies the users who are directly related to or who contribute to this data. Thus, each piece of data is associated with a possibly empty group of users. We call this group the participating group of the data. Intuitively, in most cases, the social clique concerned with a piece of data is the one that contains its participating group. As an example, consider a family photograph being shared on a social network. There are a number of family members who appear in it, and these members form the participating group of the photograph. This photograph is most likely of concern only to the members of the family, where the family is a social clique that contains the participating group of those members that appear in the picture.

Harsh Bhasin, Rohan Mahajan, “Genetic Algorithms Based Solution To Maximum Clique Problem”, ISSN: 0975-3397 Vol. 4 No. 08 Aug 2013, suggests the solution of above problem with the help of Genetic Algorithms (GAs). The work also takes into consideration, the various attempts that have been made to solve this problem and other such problems [13]. The present problem is to find out a clique with maximum cardinality. Common understanding is that it is an NP Complete problem. The point can be proved with the following example. For example, if there are 50 vertices in a graph and the number of fully connected sub graph are to be found then the total number of such graph will be

$${}^nC_2 + {}^nC_3 + {}^nC_4 + {}^nC_5 + \dots + {}^nC_{47} + {}^nC_{48} + {}^nC_{49}$$

Which is equal to $2^{50} - 1 \sim 2^{50}$

Now, a very fast computer if processes 106 instructions in 1 sec then it will take 35.7 years to elucidate all the sub-graphs and find out the maximum cliques. If the graph contains more vertices then the complexity will increase as per the problem. In such cases GAs comes to our rescue. They are known to perform efficiently if sample space is huge. Genetic algorithms imitate the process of natural selection. A population is generated which consist of chromosomes. Chromosomes are further made up of cells. In our case the cells are binary that is 0 or 1. In the implementation the pseudo random number generator of the language generates a number up to 100. If the number is less than 50 then the cell of that chromosome become 0 and if number is greater than 50 then the cell becomes 1. The numbers of cells in a chromosome depend on the problem at hand. In our case the number of cells in a chromosome is equal to number of vertices in a graph. The number of chromosome can be an optimal number large enough to contain a feasible solution.

3.1 Significance

Since the MODIFIED CLIQUE problem has important applications, designing an algorithm with good performance becomes necessary and important. A lot of algorithms for solving the MODIFIED CLIQUE problem have been proposed in the literature since the 1990's. The early work focused on the exact algorithms. Some of them achieved success on small graphs (less than 400 vertices). For example, Balas and Yu [2] improved the implicit enumeration algorithm by reducing the number of sub-problems generated from each node of the search tree, which in turn, reduced the size of the whole search tree. But still, the running time of exact algorithms increases exponentially with the size of graphs. So it is not practical to solve large problems by exact algorithms. The next step is to approximate the maximum clique size to be within a certain factor of the optimal solution. Therefore, it is necessary to solve the problem using heuristic algorithms. Heuristic algorithms cannot guarantee the quality of solutions but in practice they have been observed to produce very good solutions.

Area of research concentrates on solution of Maximum Clique Problem using a genetic algorithm and Tarjan's Algorithm which is a parallel search procedure utilizing BFS and is inspired by the mechanisms of evolution in natural systems and the scenarios under which the solutions are applicable, the various qualitative parameters satisfied by the solutions and their associated costs in order to meet the current and future resource pool of a dynamic requirements in the most efficient way based on various qualitative and quantifiable parameters, so that virtualization or multi-tenancy can be easily deployed for providing various search services.

3.2 Objective

1. To identify various qualitative and quantifiable parameters for the fitness function to make intelligent decisions regarding fitness evaluation of chromosomes in a given population.
2. To design better clique extraction method for finding Maximum Cliques.

3. To explore various Clique finding algorithms and their solutions for helping MODIFIED CLIQUE based solution to Maximum Clique Problem.

3.3 Methodology

In order to build the Genetic Algorithm based solution to Maximum Clique Problem the following methodology is followed:-

1. **Identification**

In this step the requirement analysis is done. This requires a task analysis to be done to determine the requirements, the inputs and outputs the prospective users.

2. **Conceptualization**

In this the proposed program is designed to understand and define the specific relationships and interactions in the problem domain. The key concepts, the relationships, processes and control mechanisms are determined. This is the initial stage of knowledge acquisition.

3. **Formalization**

This involves organizing the key concepts and information into formal representations i.e. rules for the Encoding and Crossover. It involves deciding the attributes to be determined to solve the problem and to build the initial mutated result.

4. **Implementation**

This involves mapping of the formalized population into a framework of the development tool (MATLAB) to build a working Matrix. The contents of matrix structures, inference rules and control strategies established in the previous stages are organized into suitable format.

3.4 Sources of Data

The first population for the matrix computation will be developed through extensive study of journals, books, white papers etc. as well as experts in the field of Advanced Data Structures, Heuristic Searching Techniques and Evolutionary Search Strategies etc.

3.5 Research Design

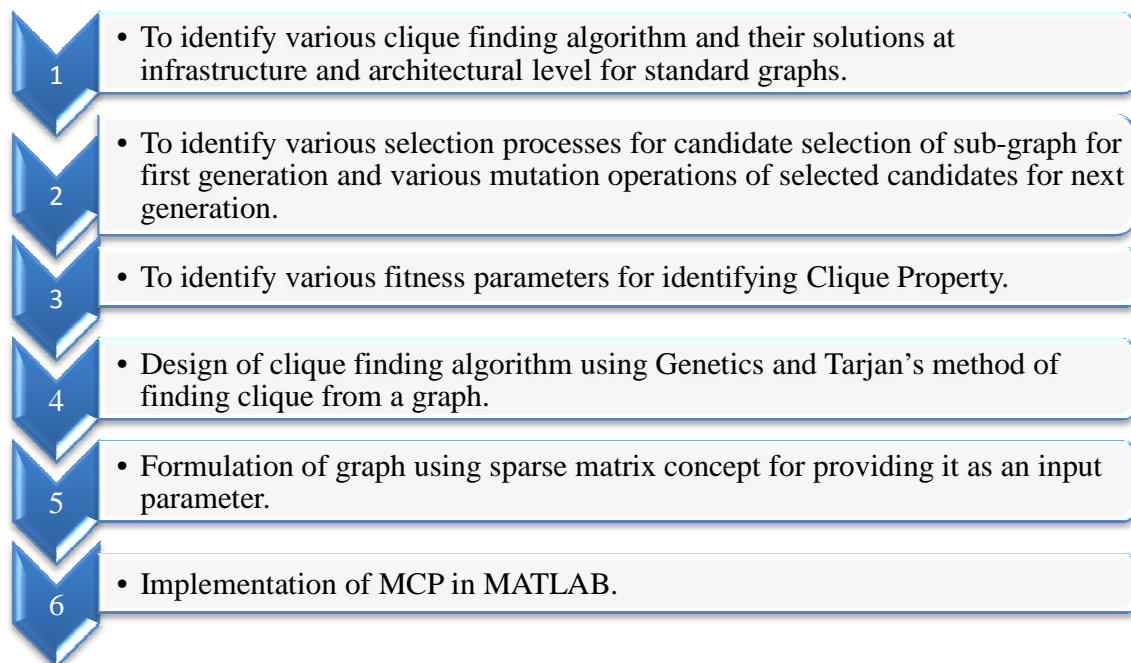







Fig. 3.1 Research Design

DISSERTATION MONTHLY PROGRESS

4.1 GANTT CHART

Task	Start	End	Duration	2014				
				Jan	Feb	Mar	Apr	May
1. To Identify various clique finding algorithm and their solutions at infrastructure and architectural level.	05/1/14	30/1/14	25					
2. To identify various selection processes for candidate selection of sub-graph for first generation and various mutation operations of selected candidates for next generation.	24/1/14	16/2/14	22					
3. To identify various fitness parameters for identifying Clique Property.	17/2/14	22/3/14	32					
4. Design of clique finding algorithm using Genetics and Tarjan's method of finding clique from a graph.	23/3/14	12/4/14	18					
5. Formulation of graph using sparse matrix	03/4/14	27/4/14	24					

concept for providing it as an input parameter.				
6. Implementation of MCP in MATLAB.	16/4/14	10/5/14	24	

Table 4.1 Gantt chart

CHAPTER 5

RESULTS AND DISCUSSIONS

5.1 Read time and Run time of Standard graphs of DIMACS

Graph Name	No. of Nodes	Edges	Read_Time (ms)	Run_Time (ms)
brock200	50	80	0.05	26.56
brock400	100	50	0.18	629.72
brock800	100	200	0.72	411.89
c-fat200	100	50	0.05	20.35
c-fat500	100	100	0.25	408.46
hamming6	30	16	0.08	5.36
hamming8	50	50	0.07	29.02
johnson8	30	50	0.02	5.89
johnson16	30	50	0.05	9.87
johnson32	50	50	0.28	116.85
keller6	50	400	17.65	11366.83
MANN_a27	30	75	0.2	667.85
MANN_a45	30	250	1.53	20452.38
p_hat300	75	75	0.1	33.59
p_hat500	100	100	0.3	690.07
p_hat700	50	150	0.63	758.36
p_hat1000	100	200	1.1	434.81
san200	30	50	0.05	57
san400	100	100	0.17	130.29

Table 5.1 Read and Run Times of Standard graphs

5.2 Results of HGAMC algorithm

Graph	Nodes	Edges	Max. Clique	HGMCA Best	Avg. Size	Avg. CPU Times
c-fat200-1	200	1534	12	12	12	0.02
c-fat500-1	500	4459	14	14	14	0.12
Johnson16-2-4	120	5460	8	8	8	0.04
Johnson32-2-4	496	107880	16	16	16	0.32
Keller-4	171	9435	11	11	11	0.05
Keller-5	776	225990	27	27	26.1	2.32
Keller-6	3361	4619898	59	54	53.2	103.4
Hamming10-2	1024	518656	512	512	512	67.69
Hamming8-2	256	31616	128	128	128	3.54
San200-0.7-1	200	13930	30	30	22.0	0.1
San400-0.5-1	400	39900	13	13	8.6	0.2
San400-0.9-1	400	71820	100	100	98.6	0.96
Sanr200-0.7	200	13868	18	18	16.7	0.08
Sanr400-0.5	400	39900	13	13	12.4	0.22
San1000	1000	250500	15	10	9.3	1.02
Brock200-1	200	14834	21	21	18.2	0.1
Brock400-1	400	59723	27	25	22.7	0.24
Brock800-1	800	207505	23	21	20.3	1.68
p-hat300-1	300	10933	8	8	8.0	0.12
p-hat500-1	500	31569	9	9	8.8	0.34
p-hat700-1	700	60999	11	11	9.5	0.54
p-hat1000-1	1000	122253	10	10	9.6	1.00
p-hat1500-1	1500	284923	12	12	10.2	2.78
MANN-a27	378	70551	126	126	123.4	0.70
MANN-a45	1035	533115	345	339	336.2	5.48

Table 5.2 Avg. CPU Elapsed times of Standard graphs

5.3 Comparison of MODIFIEDCLIQUE results with other algorithms

Algorithm Graph	HGAMC	BK (Best)	ACMCP	GAMC	MODIFIEDCLIQUE
c-fat200-1	12	12	12	12	12
c-fat500-1	14	14	14	14	14
Johnson16-2-4	8	8	8	8	8
Johnson32-2-4	16	16	16	16	16
Keller-4	11	11	11	11	11
Keller-5	27	26.4	27	26.3	27
Keller-6	54	51.88	53	51.4	59
Hamming10-2	512	512	512	512	512
Hamming8-2	128	128	128	128	128
San200-0.7-1	30	29.6	30	30	30
San400-0.5-1	13	8.6	7	9.8	13
San400-0.9-1	100	100	50	100	100
Sanr200-0.7	18	18	17	17.4	18
Sanr400-0.5	13	12.9	12	11.9	13
San1000	10	9.3	8	10.5	15
Brock200-1	21	20.3	20	18.2	21
Brock400-1	25	24.2	20	23.6	27
Brock800-1	21	20.3	18	19.2	23
p-hat300-1	8	8	8	8	8
p-hat500-1	9	9	9	9	9
p-hat700-1	11	10.4	11	10.3	11
p-hat1000-1	10	9.6	10	9.9	10
p-hat1500-1	12	11.1	11	10.4	12
MANN-a27	126	123.5	125	125	126
MANN-a45	339	336.2	337	342	345

Table 5.3 Comparison of Number of Cliques found

5.3.1 Comparison Graph of MODIFIEDCLIQUE results with other algorithms

All algorithms (HGAMC, BK, ACMCP and GAMC) show almost same results of maximum number of cliques excepting HGAMC which is an optimization of GAMC. Horizontal axis shows standard graphs and vertical axis shows Maximum Number of cliques.

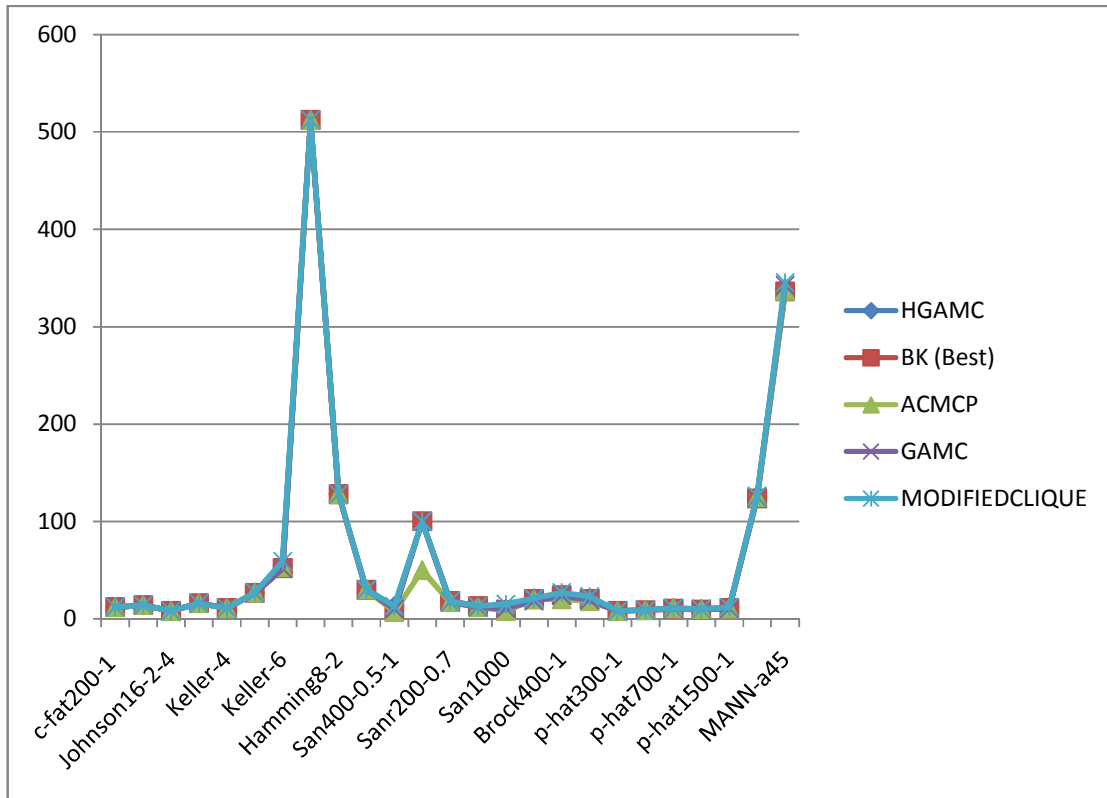


Fig. 5.1 Comparison of MODIFIEDCLIQUE number of cliques.

5.4 Comparison of MODIFIEDCLIQUE cputimes with other algorithms

Algorithm Graph	HGAMC	BK (Best)	ACMCP	GAMC	MODIFIEDCLIQUE
brock200	9.72	9.13	9.62	8.99	16.5315
brock400	17.63	17.63	17.63	17.63	119.1723
brock800	25.34	25.34	25.34	25.34	96.4398
c-fat200	11.22	11.22	11.22	11.22	120.8724
c-fat500	19.55	19.55	19.55	19.55	108.4654
hamming6	6.78	6.16	8.78	6.78	65.1356
hamming8	10.13	10.93	11.13	10.13	29.0122
johnson8	14.15	14.15	14.15	14.15	152.89
johnson16	21.33	21.33	21.33	21.33	99.1874
johnson32	26.87	27.87	27.87	27.87	96.1985
keller6	14.15	14.95	14.35	14.95	166.1083
MANN_a27	56.23	56.23	56.23	56.23	67.8512
MANN_a45	66.76	66.76	66.76	66.76	52.1038
p_hat300	41.38	43.38	43.48	43.38	133.2509
p_hat500	72.15	76.15	76.15	76.15	90.1037
p_hat700	81.37	88.29	88.29	88.29	148.0316
p_hat1000	132.31	122.13	122.30	122.39	154.1081
san200	29.14	31.70	31.70	31.70	143.4521
san400	51.16	55.69	55.69	55.69	131.9219

Table 5.4 Comparison of cputimes

5.5 Comparing Performance of various Standard graphs

Various standard graphs given under DIMACS benchmark has experimentally tested for finding maximum number of cliques in them. Standard results of all algorithms differ from results of MODIFIEDCLIQUE algorithm. Other algorithms like HGAMC, BK, ACMCP and GAMC shows almost same results of number of cliques and their CPU elapsed times excepting HGAMC which is an optimization of GAMC. MODIFIEDCLIQUE algorithm shows quite deviations in respects, CPU elapsed time as well as maximum number of cliques. Horizontal axis shows standard graphs and vertical axis shows CPU elapsed time in milliseconds.

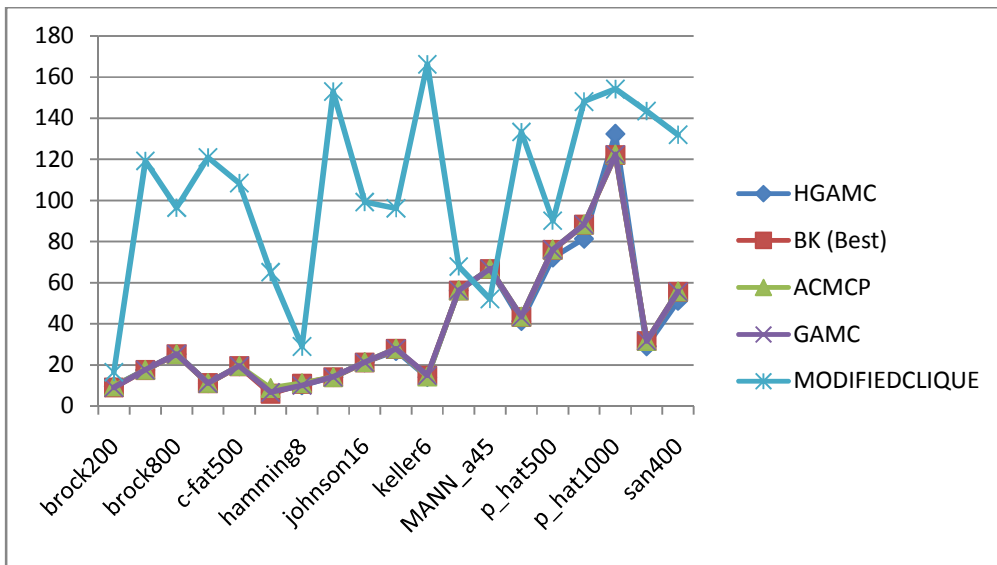


Fig. 5.2 Performance graph of various standard graphs

5.6 Viewing graph in Bio-Informatics

Input Sparse matrix is acting as an input parameter for biograph function. Viewing biograph in MATLAB by providing Sparse Matrix produces interactive output showing nodes with directed edges.

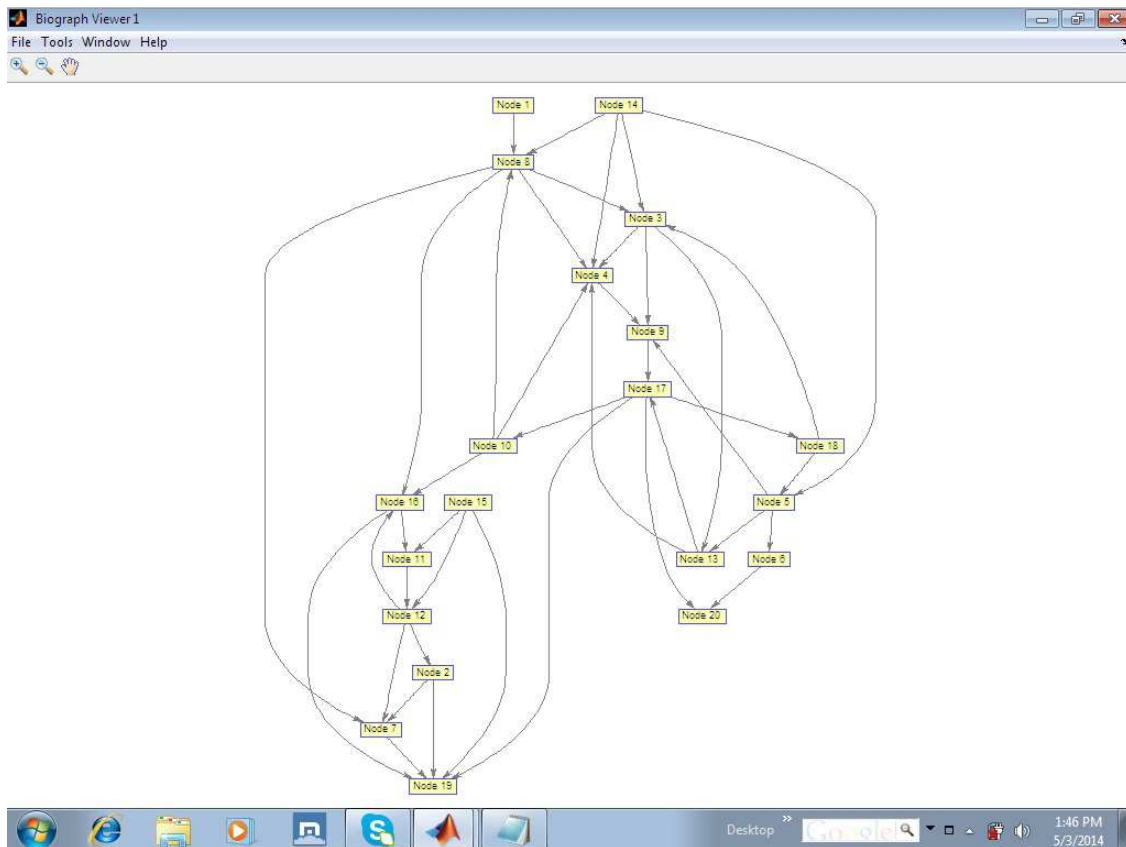


Fig. 5.4 Input graph

5.7 Realization of Input Graph

Realizing sparse matrix in MATLAB shows scattered dots representing nodes. This plot is also interactive one. Edit tools on this fig file can be used to edit this plot. Corresponding data set at the backend will be changed by performing some edit work on fig file.

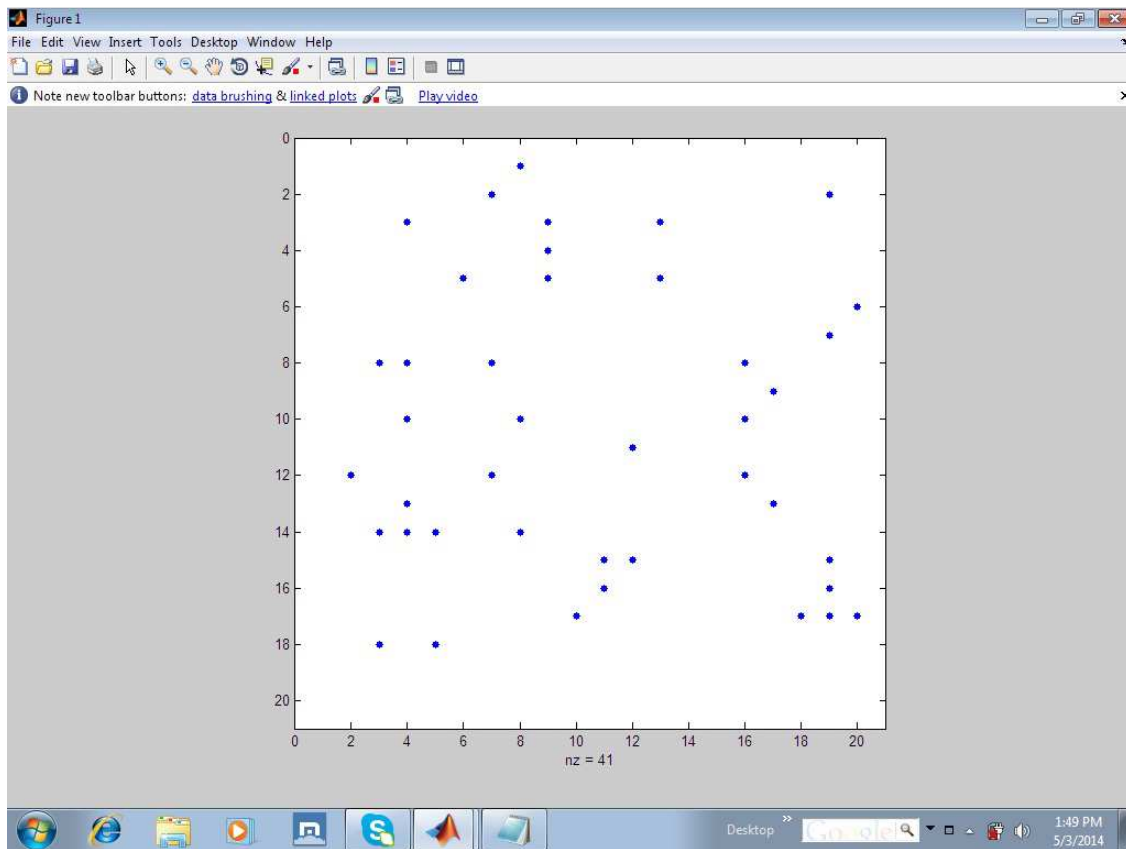


Fig. 5.5 Plot of Adjacency Sparse Matrix

5.8 Designing interface for Modified Clique finding Approach

Interface for simulating MODIFIEDCLIQUE method contains three input parameters. Two input vectors and one size of sparse matrix. These three input vectors are represented with white colored text boxes. Another three are output parameters represented with different color (pink) in this interface. One output parameter gives CPU elapsed time for finding Maximum Cliques using MODIFIEDCLIQUE algorithm. Second output parameter gives status of each node that each node is a part of some Clique. While third output parameter gives Maximum Number of Cliques found in the input Graph.

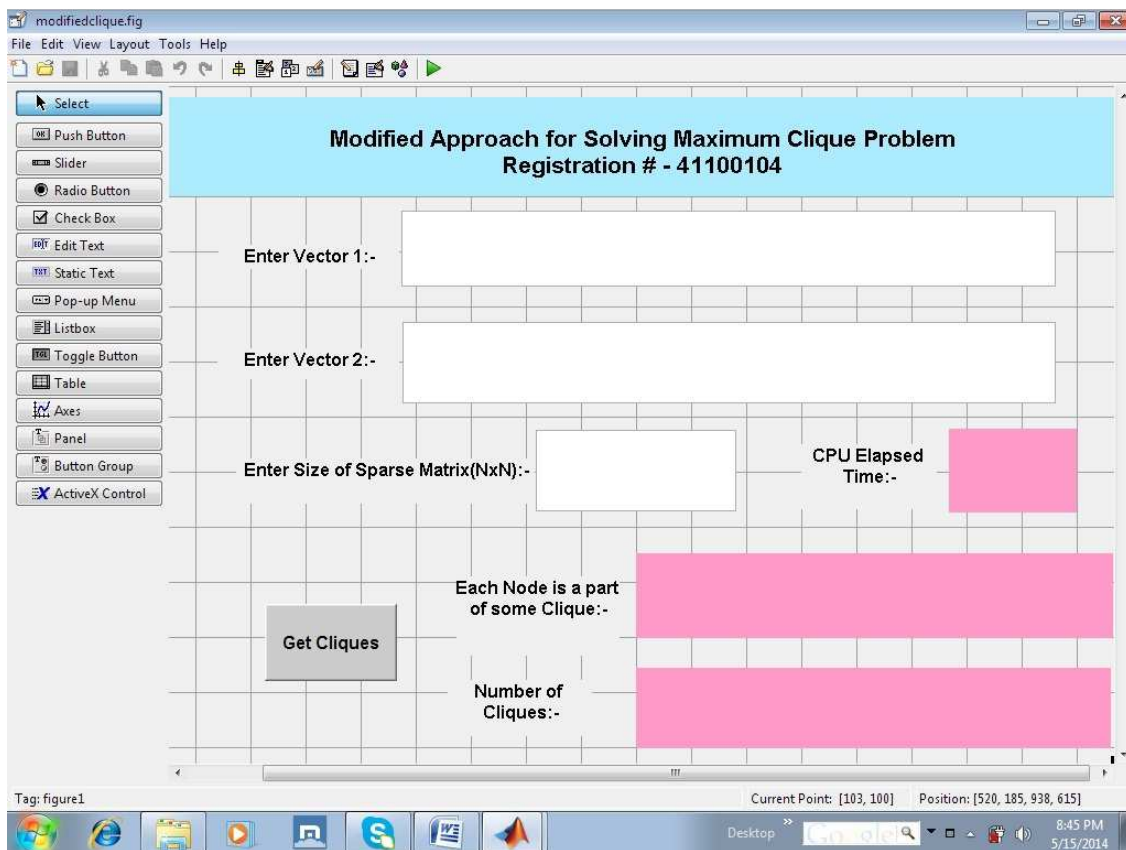


Fig. 5.6 Fig file of Interface Design

5.9 Implementation of MODIFIEDCLIQUE method

Two input vectors gives connectivity of vertices of graph. Means first elements of input vector 1 and vector 2 represents an edge. Here in interface first element of input vector 1 is 14th node and first element of input vector 2 is 4th node of graph. This represents an edge between 14th and 4th node that is reflected in Fig. 12. Third input parameter that is size of input matrix is making MATLAB to squeeze this input matrix 20x20 to its corresponding sparse matrix.

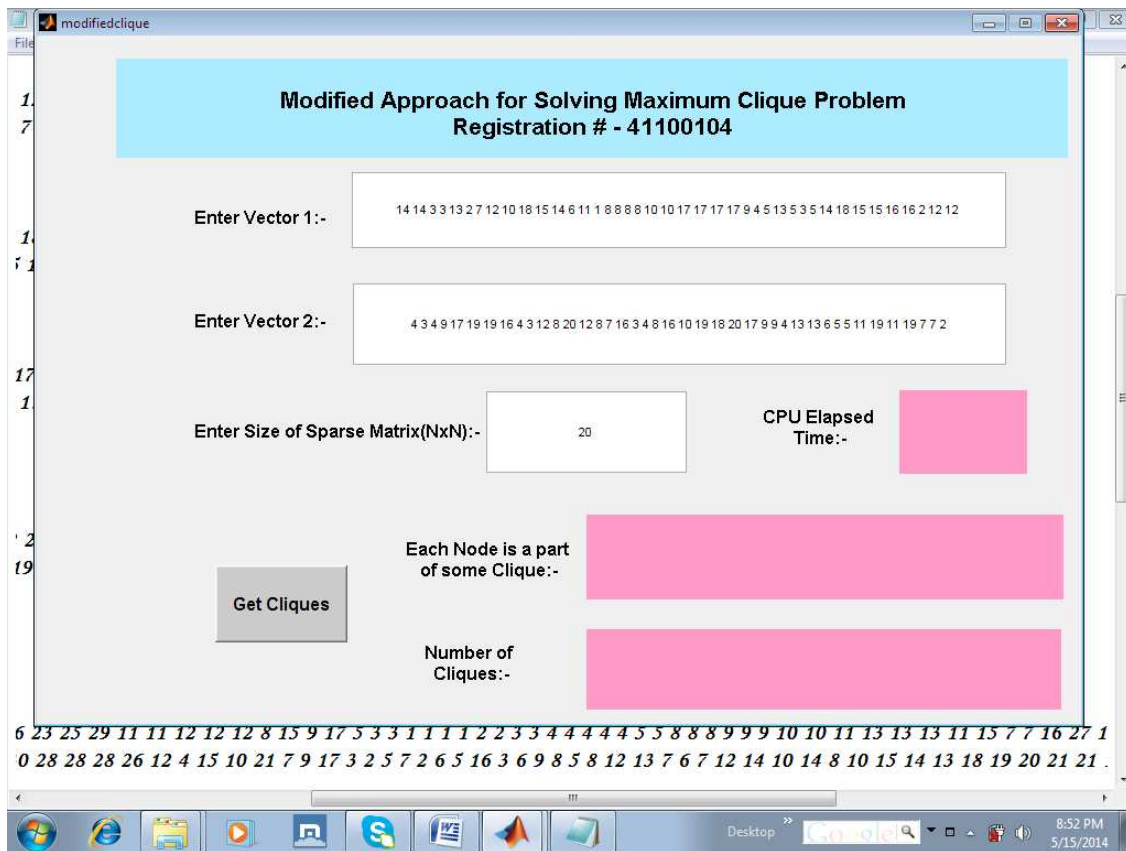


Fig. 5.7 Interface MODIFIEDCLIQUE

5.10 Result containing Maximum Cliques

Output graph is directed graph representing cliques with different color codes. Here in Fig. 12 MODIFIED CLIQUE algorithm has found Maximum 10 cliques. Largest clique is coded with light green color.

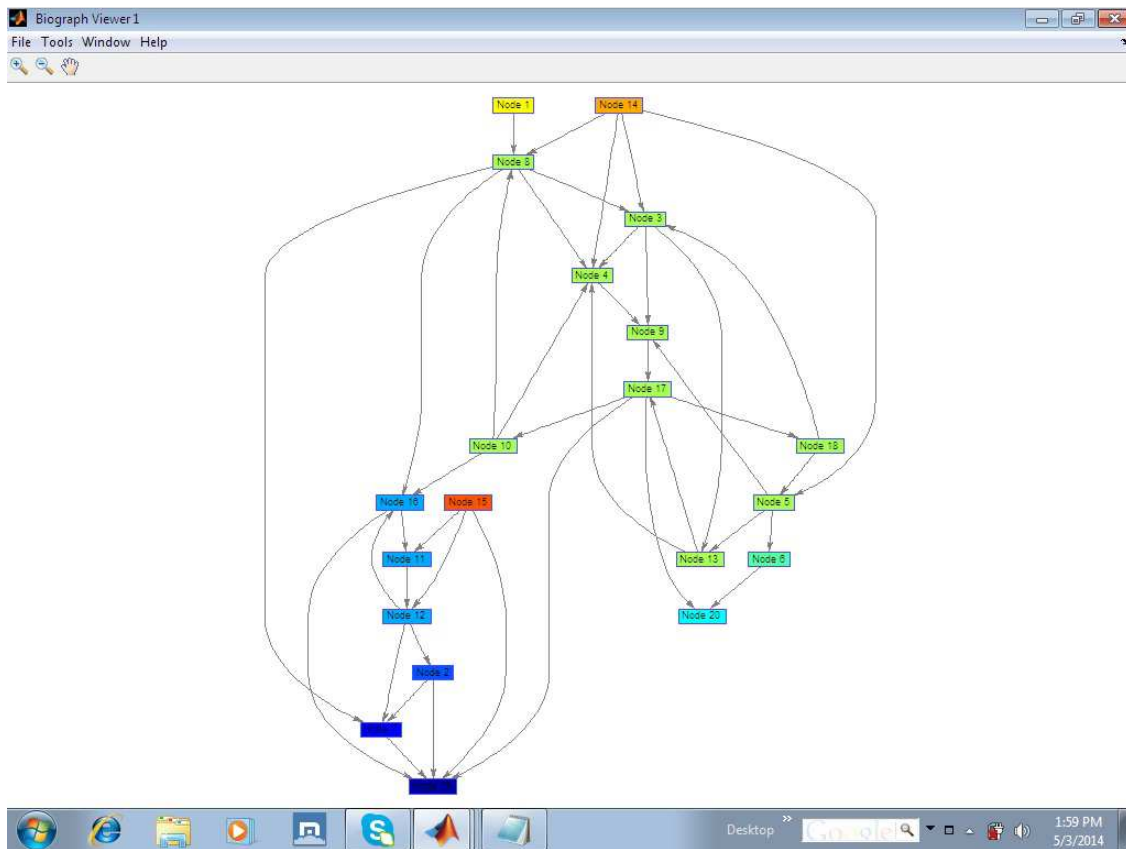


Fig. 5.8 Maximum Cliques

5.11 Result showing each node's containability, No. of components and CPU elapsed time

Results of standard graph Hamming20 are shown here in Fig. 13. CPU elapsed time of MODIFIEDCLIQUE algorithm is quite close to results of other algorithms. Second output parameter shows “Each Node is a part of some Clique” contains first element as 8 which means that first node is a part of 8th Clique out of total 10 Cliques found.

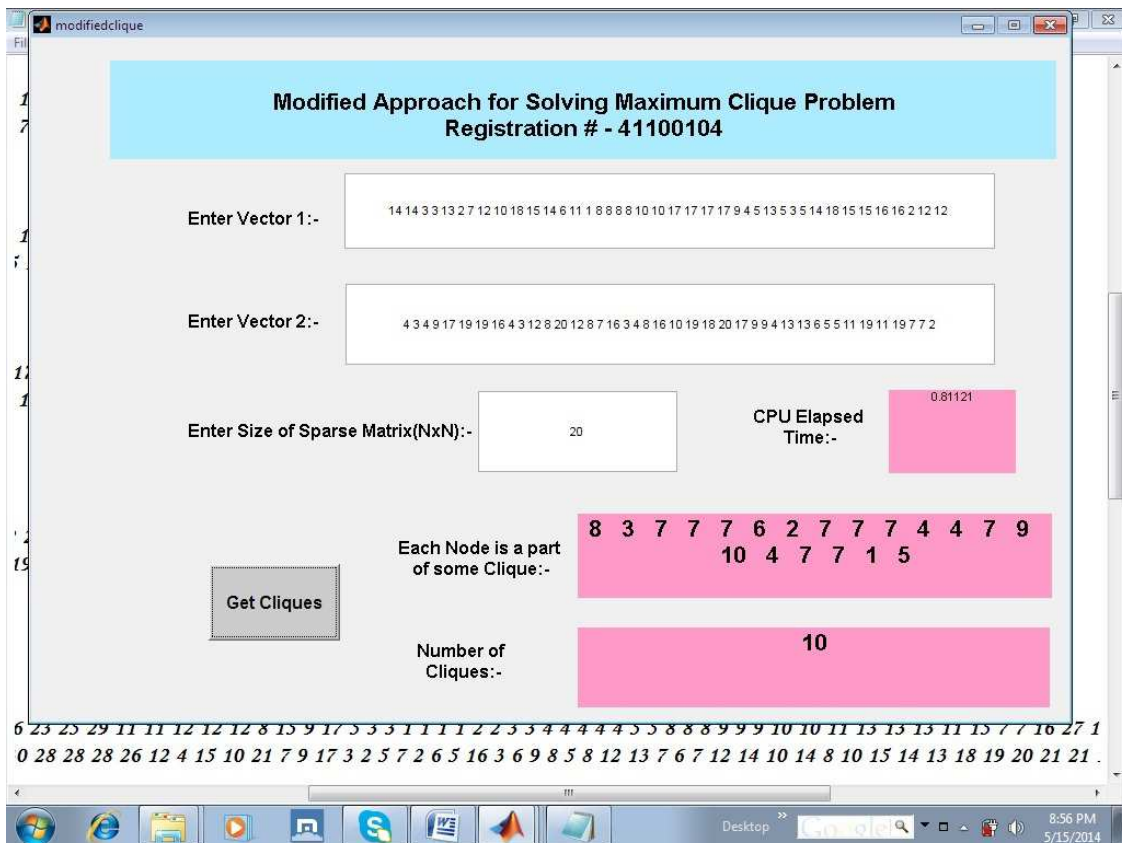


Fig. 5.9 Listing Cliques with their number

5.12 Input Vectors on Command Window

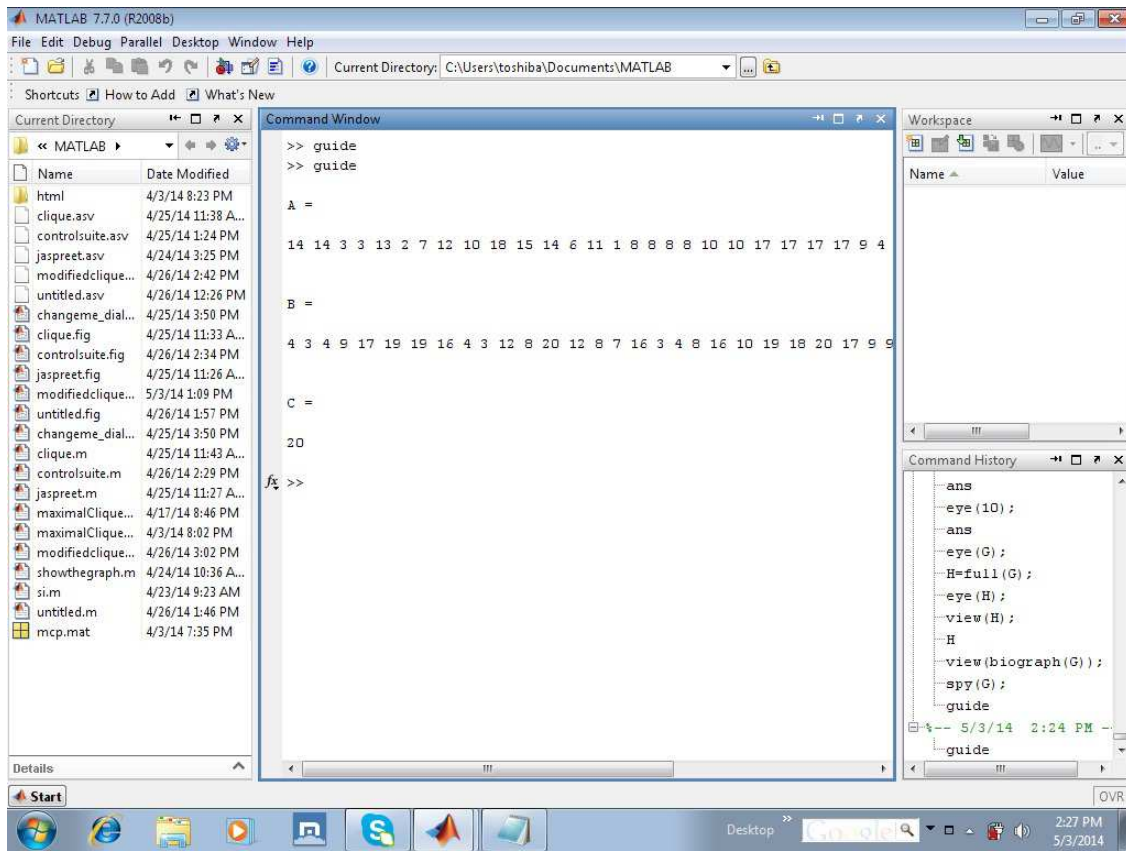


Fig. 5.10 Vectors on Command Window

5.13 Command window showing S and C

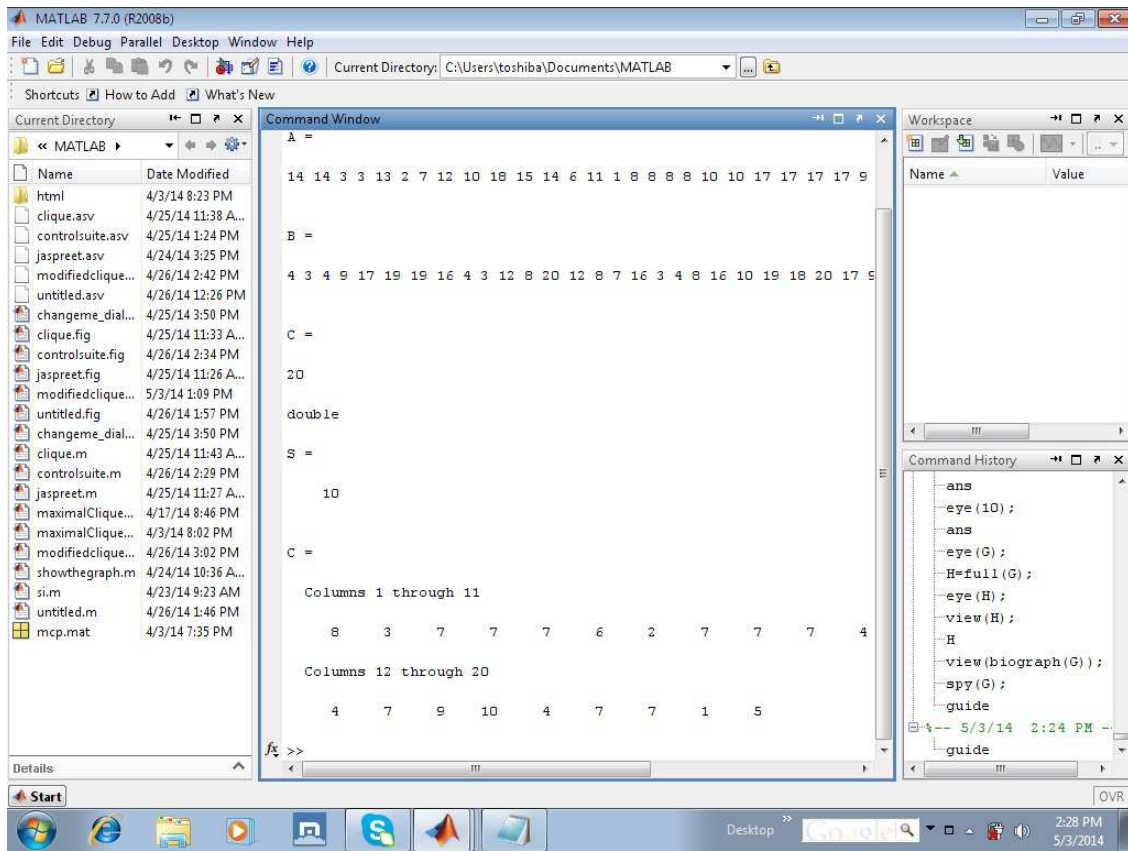


Fig. 5.11 Results in Command window

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

The method MODIFIEDCLIQUE has performed well only on standard graphs like brock-20 and keller-6. The main feature which is stressed in MODIFIEDCLIQUE is graph preprocessing. Algorithm MODIFIEDCLIQUE is tested on DIMACS benchmark graphs and it performs moderately, not that much well like ACMCP, BK and HGAMC. MODIFIEDCLIQUE has performed relatively poor on hamming-20 graph as compared to HGAMC. HGAMC is optimization of GAMC. Its standard results are equivalently good as GAMC.

Future work could consider changing the termination condition in MODIFIEDCLIQUE method. If the method reaches stagnancy, then it can reallocate vectors for exploration of new cliques that can be extended from already calculated ones. For exploration of maximum number of cliques in input graph MODIFIEDCLIQUE can drop existing vectors and restart new search for more cliques in the new start. In this way the algorithm has a chance to escape from local optimization and explore more of search space. However one needs to find a good reallocation method.

CHAPTER 7

REFERENCES

- [1] Tsukiyama S., Ide M., Ariyoshi H. and Shirakawa I., “A new algorithm for generating all the maximal independent sets” *SIAM J. Comput.*, 6 (1977) 505–517.
- [2] M.R. Garey, D.S. Johnson, *Computers and Intractability. “A Guide to the Theory of NP-completeness”* Freeman, New York, 1979.
- [3] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, “Optimization by Simulated Annealing” *Science*, 220, 1983, pp. 671-680.
- [4] E. Balas and C. S. Yu, “Finding a Maximum Clique in an Arbitrary Graph” *SIAM J. Comput.*, 14, 1986, pp. 1054-1068.
- [5] D. E. Goldberg, “Genetic Algorithms in Search, Optimization and Machine Learning” Addison-Wesley, Boston, MA, 1989.
- [6] P. Berman and A. Pelc, “Distributed Fault Diagnosis for Multiprocessor Systems” *Proc. of the 20th Annual Int. Symp. On Fault-Tolerant Computing (Newcastle, UK)*, 1990, pp. 340-346.
- [7] J.Hertz, A. Krogh and R. G. Palmer, “Introduction to the Theory of Neural Computation” Assison-Wesley, Redwood City, CA, 1991.
- [8] P. M. Pardalos and J. Xue. “The Maximum Clique Problem” *Journal of Global Optimization*, 4, 1994, pp. 301-328.
- [9] P.M. Pardalos, J. Xue, “The maximum clique problem” *J. Glob. Optim.* 4 (1994) 301–328.
- [10] R. Battiti and M. Protasi, “Reactive Local Search for the Maximum Clique Problem” Technical Report TR-95-052, International Computer Science Institute, Berkeley, CA, 1995.
- [11] D. S. Johnson and M. A. Trick (eds.), “Cliques Coloring and Satisfiability, Second DIMACS Implementation Challenge” DIMACS 26, American Mathematical Society, 1996 (see also <http://dimacs.rutgers.edu/Volumes/Vol26.html>).

- [12] Avis D. and Fukuda K., “Reverse search for enumeration” *Discrete App. Math.*, 65 (1996) 21–46.
- [13] David R. Wood, “An algorithm for Finding a maximum clique in a graph” 1997 Elsevier Science B.V. PII S0167-6377(97)00054-0
- [14] Johnson D. S., Yanakakis M. and Papadimitriou C. H., “On generating all maximal independent sets” *Info. Proc. Lett.*, 27 (1998) 119–123.
- [15] Kumar S. R, Raghavan P., Rajagopalan S., and Tomkins A., “Trawling the web for emerging cyber-communities” Proc. the Eighth International World Wide Web Conference, Toronto, Canada, 1999.
- [16] Patric R. J. Ostergard, “A Fast Algorithm for the maximum clique problem”, 2002 Elsevier Science B.V. PII: S0166-218X (01)00290-6
- [17] Dominic W. and Beate D. “A graph model for unsupervised lexical acquisition”. Proceedings of the 19th International Conference on Computational linguistics(2002). ACL: Morristown, USA.
- [18] Volker S. “Finding All Maximal Cliques in Dynamic Graphs”. Computational Optimization and Applications (2004) Vol. 7, Issue 2. Kluwer Academic Publishers: Norwell, USA.
- [19] R.Rama, Suresh Badarla and Kamala krithivasan, “Clique-detection algorithm using clique-self-assembly”, IEEE DOI 10.1109/BIC-TA.2011.32
- [20] Hakan Y. and Christopher K. “Detecting Social Cliques for Automated Privacy Control in Online Social Networks”, Fourth International Workshop on Security and Social Networking, Lugano (19 March 2012).
- [21] C. Bron and J Kerbosch, “Bron Kerbosch Algorithm” PopulPublishing ©2012 Page-96, ISBN-6136404559 9786136404554
- [22] Harsh Bhasin, Rohan Mahajan, “Genetic Algorithms Based Solution To Maximum Clique Problem”, ISSN: 0975-3397 Vol. 4 No. 08 Aug 2013

8.1 Glossary of Terms

A

ACMCP Ant Colonization technique for Maximum Clique Problems

ANN Artificial Neural Networks

B

Bipartite cliques

BK Bron Kerbosch algorithm'

Boltzmann selection

brock200

brock400

brock800

C

c-fat200

c-fat500

Chromosomes

Clique

Crossover

D

DIMACS

DIMACS Discrete Mathematics and theoretical Computer Science

DNA

F

Fitness Function

G

GA Genetic Algorithm

GAMC Genetic Algorithm for finding Maximum Cliques

Genetic Algorithm

GS Graph Structure with Clique

H

hamming6

hamming8

HGAMC Hybrid Genetic Algorithm for finding Maximum Cliques

J

johnson16

johnson32

johnson8

K

keller6

KP Knapsack Problem

M

MANN_a27

MANN_a45

MCP

MCP Maximum Clique Problem

MODIFIEDCLIQUE

Mutation

N

Natural Evolution

NP Complete

NP Hard

NP Non Polynomial complexity

P

p_hat1000

p_hat300

p_hat500

p_hat700

Population

R

Rank selection

Reverse Search

Roulette wheel selection

S

san200

san400

Simulated annealing

Steady state selection

T

Tabu Search

Tarjan's Algorithm

Tournament selection

TSP Travelling Salesmen Problem

8.2 Abbreviation

ACMCP Ant Colonization technique for Maximum Clique Problems

ANN Artificial Neural Networks

BK Bron Kerbosch algorithm'

DIMACS Discrete Mathematics and theoretical Computer Science

GA Genetic Algorithm

GAMC Genetic Algorithm for finding Maximum Cliques

GS Graph Structure with Clique

HGAMC Hybrid Genetic Algorithm for finding Maximum Cliques

KP Knapsack Problem

MCP Maximum Clique Problem

NP Non Polynomial complexity

TSP Travelling Salesmen Problem