**The Pennsylvania State University**
**The Graduate School**
**Capital College**

# Finding Maximum Clique with a Genetic Algorithm

A Master's Paper in
Computer Science
by
Bo Huang



Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

April 2002

## Abstract

This paper presents a hybrid genetic algorithm for the maximum clique problem. The main features of the algorithm are a strong local optimizer to accelerate the convergence and the adaptive genetic operators to fine tune the algorithm. Our algorithm was tested on DIMACS benchmark graphs of size up to 3,361 vertices and up to 4,619,898 edges. For more than 92% of 64 graphs in the test set, our algorithm finds the maximum clique, and for a small number of graphs, the results achieved by our algorithm are almost as large as the best-known results found by other algorithms. This result is comparable to one of the best existing algorithms for the maximum clique problem.

# Table of Contents

## Acknowledgement

I would like to thank Dr. Thang N. Bui for his valuable suggestions and constructive criticism. I couldn't achieve the success of this project and the clarity of this paper without his guidance. I am grateful to the committee members, Dr. P. Naumov and Dr. L. Null, for reviewing this paper. I also appreciate Mr. P. Eppley and Mr. H. Royer's help.

# List of Figures

# List of Tables

# 1 Introduction

Let $G = (V, E)$ be an undirected graph on $n$ vertices, where $V$ is the vertex set and $E$ is the edge set. A *clique* in $G$ is a subgraph of $G$ in which there is an edge between any two vertices. The *size* of a clique is the number of vertices in the clique. The *maximum clique* problem (MAXCLIQUE) is to find the largest clique in a given graph $G$.

The MAXCLIQUE problem is an $NP$-hard problem [24], which means that unless $P = NP$, there is no polynomial time algorithm for solving MAXCLIQUE. Furthermore, unless $P = NP$, there is no polynomial time algorithm that can approximate the maximum clique size to within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$, where $n$ is the number of vertices [12]. Since $P = NP$ is widely believed to be false, it is unlikely that we can solve the problem in polynomial time using exact or approximation algorithms. Therefore, using polynomial time heuristic algorithms without performance guarantees becomes a practical way to solve MAXCLIQUE. A lot of heuristic algorithms have been proposed for MAXCLIQUE such as greedy heuristics, simulated annealing, neural network and tabu search [24].

The MAXCLIQUE problem has many real world applications such as coding theory, fault diagnosis, printed circuit board testing and computer vision [24]. A collection of test graphs derived from those problems has been made available for the comparative study of the performance of algorithms for MAXCLIQUE [15].

*Genetic algorithms* (GAs) are a family of heuristic algorithms inspired by the evolution in the natural world [14]. These algorithms encode potential solutions, called chromosomes, then apply recombination and mutation operators to these chromosomes to evolve good solutions. GAs have been successfully applied to many $NP$-hard problems in different areas.

In this paper, a hybrid genetic algorithm is given to solve the MAXCLIQUE problem. The algorithm is a genetic algorithm combined with a local optimizer and graph preprocessing. The algorithm was tested with nine classes of graphs ranging in size from 200 to 3,361 vertices and up to

4,619,898 edges. It produced very good results compared with other well known heuristic algorithms. In most cases it can find the maximum clique. For a few classes of graphs, the best result it generated is very close to the size of known maximum clique.

The rest of this paper is organized as follows. Section 2 introduces several important applications of the MAXCLIQUE problem and other heuristic algorithms that have been used to solve the problem. Section 3 describes the principles of genetic algorithms and problems to which they have been applied. In Section 4, we give a hybrid genetic algorithm for solving the MAXCLIQUE problem. Section 5 shows the test results of this algorithm and a comparison with other well known algorithms. Conclusions are given in Section 6.

# 2 The Maximum Clique Problem

In this section, we introduce several important applications of the MAXCLIQUE problem and three heuristics that have been successfully applied to the MAXCLIQUE problem.

## 2.1 Applications

The MAXCLIQUE Problem has many real world applications. It is encountered in many different fields in which either the underlying problem can be formulated as the MAXCLIQUE problem or finding the maximum clique is a precondition of solving the problem. Based on those applications, a collection of very diversified test graphs for the MAXCLIQUE problem has been created for evaluating the performance of algorithms for the MAXCLIQUE 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.

### 2.1.1 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 (for details, see [6] [25]). 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. If the Hamming distance between any two vectors in a binary code $B$ is greater than or equal to $d$, then $B$ is able to correct $\lfloor \frac{d-1}{2} \rfloor$ errors [19]. Define a *Hamming graph* $H(n, d)$ as a graph having $2^n$ vertices, $2^{n-1} \sum_{i=d}^{n} \binom{n}{i}$ edges and the degree of each vertex is $\sum_{i=d}^{n} \binom{n}{i}$. Consider all possible binary vectors of size $n$ as the vertex set and there is an edge between two vertices only if their Hamming distance is at least $d$. This problem can be formulated as a Hamming graph $H(n, d)$. 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 $\lfloor \frac{d-1}{2} \rfloor$ errors.

### 2.1.2 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 [4], the system is represented by an undirected graph $G = (V, E)$ whose vertices are processors and where edges are communication links. $G$ is defined as follows. For a given parameter $c$, let

$$k = \left\lfloor \frac{|V|}{c \log |V|} \right\rfloor$$

and let $W_0, \ldots, W_{k-1}$ be a partition of $V$ such that $c \log |V| \leq |W_i| \leq 1 + \lceil c \log |V| \rceil$ for $i = 0, 1, \ldots, k-1$. For $u \in W_i$ and $v \in W_j$ we have $(u, v) \in E$ if and only if $u \neq v$ and $|i - j| \in \{0, 1, k-1\}$. By the definition, a graph of size $n$ has $O(n \log n)$ edges. This type of graph is called a *C-FAT ring*. A distributed diagnosis is performed on this graph. The major step in the

algorithm is finding the maximum clique based on the communication links between vertices built by the results of test messages. It is assumed a test by a fault-free unit on a faulty one does not detects a fault with probability $q$, and one fault-free unit never finds another fault-free unit faulty. Therefore, if a vertex is in the maximum clique, mark it as "fault-free". If not, mark it as "faulty".

### 2.1.3 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. This is an example of a *geometric* clique problem and has been studied by Yu, Goldschmidt and Chen [26] and Yu, Kouvelis and Luo [27].

## 2.2 Algorithms

Since the MAXCLIQUE problem has important applications, designing an algorithm with good performance becomes necessary and important. A lot of algorithms for solving the MAXCLIQUE problem have been proposed in the literature since the 1950'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 subproblems 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. However, it has been proven that, unless

$P = NP$, there is no polynomial time algorithm that can approximate the maximum clique size to within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$, where $n$ is the number of vertices [12]. Since $P \neq NP$ is widely believed to be true, it is unlikely to find a good approximation for MAXCLIQUE. 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. In this section, we will introduce a number of successful heuristic algorithms besides genetic algorithms since we will discuss them in the next section.

### 2.2.1 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 [17]. 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 $\Delta f$ is the difference of the cost function between the new state and the current state, and $\tau$ is a parameter usually called the *temperature* in analogy with physical annealing, which is varied during the optimization process [24]. The simulated annealing heuristic has been ranked among one of the best heuristics for the MAXCLIQUE problem at the 1993 DIMACS challenges [15].

### 2.2.2 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 [13]. 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 MAXCLIQUE problem using neural networks. They encode the problem in different models, but most of them are based on the Hopfield model [13] 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 [24].

### 2.2.3 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 [24]. 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 MAXCLIQUE 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 [15].

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.

# 3 Genetic Algorithms

In this section, we introduce the terminologies and principles of GAs. Then we briefly discuss a number of applications to which GAs have been applied.

## 3.1 GA Introduction

A *genetic algorithm* is a parallel search procedure inspired by the mechanisms of evolution in natural systems [14]. The basic elements of GAs are problem encoding, selection methods and genetic operators such as crossover and mutation.

- **Problem encoding**: It is the way in which the chromosomes and their genes represent potential problem solutions. Chromosomes are the simulated organisms in a GA's population and genes are simple structures contained in chromosomes. Most GA applications use fixed-length, fixed-order strings to encode the potential solutions. The encoding methods usually depend on the natural properties of the problem. For example, in graph theoretic problems, binary encoding (i.e., bit strings) is the most common encoding. Other encoding methods such as many-

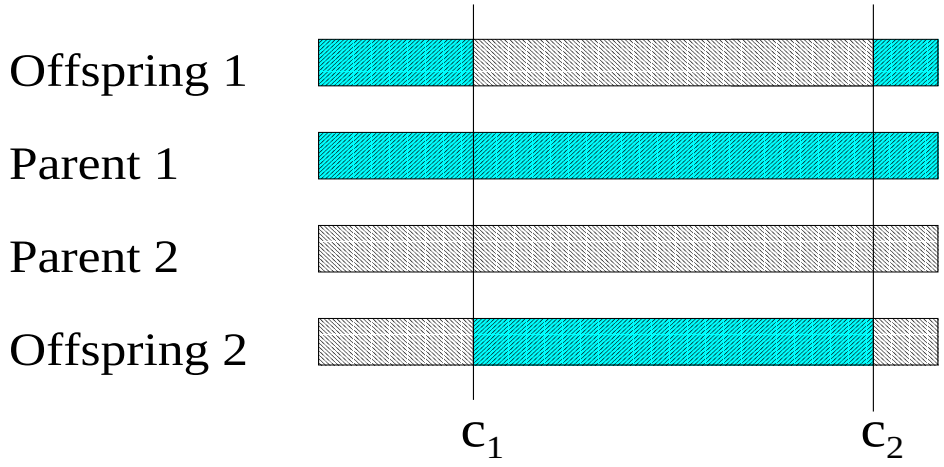


**Figure 1:** A 2-point crossover

character and real-valued encoding are used for representing condition sets and neural-network weights.

- **Selection Methods**: Selection methods are used to decide how to choose the individuals in the population that will create offspring and how to replace the old population with the new population. The purpose of selection is to emphasize the fitter individuals in the population, but it also has to avoid losing diversity of the population.

- **Crossover**: Crossover is the major instrument of variation and innovation in GAs. The simplest form of crossover is single-point crossover. It chooses a single crossover position at random and the parts of two parents after the crossover position are exchanged to form two offspring [22]. For very long chromosomes, a multipoint crossover is often used to strengthen the power of crossover. An $n$-point crossover chooses $n$ positions at random from 1 to $n$, and the segments between position $2i$ and $2i+1$ of two parents are swapped where $i = 1, 2, \ldots, \lfloor \frac{n}{2} \rfloor$. An example of a 2-point crossover is shown in Figure 1.

- **Mutation**: A mutation operator randomly changes each character in a selected chromosome into another random character with probability

$P_m$. The primary effect of mutation is to introduce new alleles, the values of genes, into the evolution process to avoid permanent fixed kinds of alleles in a population.

A general genetic algorithm scheme is shown in Figure 2, where $P(t)$ denotes the population at time $t$. The algorithm starts with a randomly generated population of potential solutions. It then goes into a simulated evolution process until the termination condition is met. The process is as follows. Select a pair of parent chromosomes from the current population according to the selection method. Apply the crossover and mutation operators on the parents to generate offspring. Evaluate the offspring using the fitness function. Replace the current population with the new population by the replacement method. After the evolution is terminated, return the best member of the population. The performance of GAs depends highly on details such as the encoding method, the genetic operators, the parameter settings and the particular criteria for success.

## 3.2 GA Applications

GAs have been used in a large number of scientific and engineering problems and models. Some examples are briefly introduced below [22].

- **Optimization**: GAs have been used in a variety of optimization tasks, including numerical optimization and combinatorial optimization problems such as the traveling salesman problem and the graph partitioning problem.

- **Automatic programming**: GAs have been used to evolve computer programming for specific tasks, and to design other computational structures such as cellular automata and sorting networks. This technique becomes a branch of evolutionary computation area, called genetic programming.

- **Machine learning**: GAs have been used for many machine learning applications, including classification and prediction tasks, such as the

```
t ← 0
Initialize a population P(t)
while (termination condition is not met) do
    Select a subset Q(t) from P(t)
    Apply crossover and/or mutation operators
       on Q(t) to obtain Q'(t)
    Evaluate Q'(t)
    Select and replace from Q'(t)∪P(t)
       to obtain P(t+1)
    t ← t+1
endwhile
return the best member of P(t).
```

**Figure 2:** A general genetic algorithm

prediction of weather or protein structure. GAs have also been used to evolve particular machine learning systems, such as weights for neural networks, rules for learning classifier systems and sensors for robots.

- **Social systems**: GAs have been used to study evolutionary aspects of social systems, such as the evolution of social behavior in insect colonies, and more generally, the evolution of cooperation and communication in multi-agent systems.

# 4 A Hybrid Genetic Algorithm for MAXCLIQUE

In this section, we present a Hybrid Genetic Algorithm for solving the Maximum Clique problem (HGAMC). The HGAMC basically follows the steps of a steady-state genetic algorithm, which means only a portion of the population is replaced at each generation. If the whole population is replaced at each generation, it is called a generational GA. The main features of the HGAMC are a strong local optimizer to accelerate the convergence and the adaptive genetic operators to fine tune the algorithm. Also, graph preprocessing is used before creating the initial population to help keep potentially good partial solutions from being disrupted by crossover. The structure of the HGAMC is shown in Figure 3. This algorithm is a modification of Bui and Eppley's genetic algorithm for solving the MAXCLIQUE problem proposed in 1995 [7]. We preserved the good features of their algorithm such as preprocessing the input graph and using local optimization, but modified the local optimization algorithm, the genetic operators such as crossover and mutation to make them adaptive. Also, we improved the methods of extracting a clique and enlarging a clique by randomizing the process of deleting or adding vertices.

The following subsections discuss the details of the problem encoding, preprocessing, initial population creation, parent selection, genetic operators, local optimization, replacement and termination condition.

**Input**: $G = (V, E)$
**Output**: A maximal clique in $G$

```
Preprocess the input graph
Create an initial population
Apply the local optimization to each chromosome
while (stopping condition is not met) do
    Select two parents, P1 and P2, from the population
    Generate two offspring by crossing over P1 and P2
    Mutate the two offspring
    Local optimize the two offspring
    Replace a population member with a better offspring
    Update stopping condition
endwhile
return the best member of the population
```

**Figure 3:** The structure of HGAMC

## 4.1 Problem Encoding

Since a potential solution of the MAXCLIQUE problem on graph $G = (V, E)$ can be its subgraph $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$, it is natural to represent a subgraph by a 0-1 vector of size $|V|$. Let $X$ be a 0-1 vector representing $G'$. Assuming all the vertices in $G$ are labeled with indices $0, 1, 2, \ldots, |V| - 1$, we assign 1 to $X[i]$ if vertex $i$ is in $G'$, otherwise assign 0 to $X[i]$. This encoding method is simple and straightforward. Also, the crossover and mutation operators will not destroy the validation of the solution (note that each chromosome is not necessarily a clique). However, with this encoding method, the performance of the algorithm is affected by the way that the vertices are labeled.

## 4.2 Preprocessing

We use a preprocessing algorithm to reorder the vertices in a chromosome. The initial encoding uses the order of the vertices as given by the input graph. The preprocessing algorithm sorts the vertices in decreasing order of their degrees. After preprocessing, the highest degree vertex should be at position 0 on the chromosome, and the lowest degree vertex should be at position $n - 1$ where $n$ is the number of vertices in the graph. The purpose of this preprocessing algorithm is to place vertices that have higher degrees closer to each other in the chromosome. It works on the assumption that vertices having higher degrees are likely to be part of a larger clique. Placing them together avoids a good part of the solution being disrupted by a multipoint crossover operator.

## 4.3 Initial Population

The initial population is created by a greedy algorithm. This algorithm generates each chromosome in the initial population as follows. First pick up a vertex at random, say $v_i$, and put $v_i$ in a subset $A$. Then randomly choose a vertex $v_j$ among the remainder of $v_i$'s adjacency list. If $v_j$ is connected with all the vertices in $A$, put it in $A$ and mark $v_j$ as "used". Repeat this step until

all the vertices in $v_i$'s adjacency list are marked as "used". Finally, assign 1 to the positions corresponding to the vertices in $A$. The result is actually a clique contained in the input graph. To diversify the initial population, we apply a mutation operator to each member. Each member is selected for mutation with a probability *initial_select_prob* and each gene of the selected chromosomes is mutated with a probability *initial_mutate_prob*. We set the *initial_select_prob* to be 0.4 and the *initial_mutate_prob* to be 0.7, since the experimental results showed that those values yield best results.

## 4.4 Parent Selection

At each generation, two parents are selected from the current population using a roulette wheel scheme. Under this scheme, the probability of a chromosome being selected is proportional to its fitness. But this method could cause the population to become homogeneous too quickly if the fitness values vary a lot. That is, the exceptionally good members have a very high probability of being selected at each generation, thus the population will become similar to those good members and lose diversity. To avoid this problem, we scale the fitness before applying the selection algorithm to make sure that the difference between the best fitness value and the worst fitness value is in a certain range. The scaled fitness values are only used for selecting parents. That is, we don't really change the fitness value of each chromosome.

## 4.5 Genetic Operators

### 4.5.1 The Fitness Function

Since a chromosome must be a clique when its fitness is evaluated, we use the size of the clique, i.e., the number of 1s in a chromosome as its fitness. This is very easy to compute. So the cost of the algorithm is greatly reduced comparing to those algorithms that use sophisticated fitness function, since computing fitness is needed to be done in each generation for all changed chromosomes.

### 4.5.2 Crossover

A multipoint crossover operator is used in this algorithm. The number of cut points is changed during the process of evolution. In the beginning, we intend to set it higher to improve the diversity of the population. Towards the end, the diversification should be less disruptive since the chromosomes contain more partial solutions. In this algorithm, we set the starting number of cut points to 10 and reduce it eventually to 2. The number of cut points is reduced by 2 every 20 generations.

### 4.5.3 Mutation

The two offspring generated by crossover are selected for mutation with probability *offspring_selection_prob*. If an offspring is selected, it is mutated with a probability *offspring_mutation_prob*. Mutating a gene is simply changing it into 1 if it is 0 and vice versa. The mutation operator is also adaptive, as it changes as the algorithm progresses. A high mutation value of 0.5 is set in the beginning of the process. It is decreased by 0.05 every 20 generations, until it reaches the minimum value of 0.05.

## 4.6 Local Optimization

A strong local optimization step is used after crossover and mutation operators generate each new offspring, and it plays a very important role in this algorithm. It attempts to find a clique contained in a chromosome as large as possible, but may not be the largest clique. The purpose of the local optimization is to speed up the convergence. There are two steps in the local optimization: clique extraction and clique improvement.

### 4.6.1 Clique Extraction

The clique extraction algorithm is based on the greedy algorithm, but we randomized the method, which is different from Bui and Eppley's algorithm. Instead of always deleting the last smallest degree vertex, we delete a vertex

```
Input: A subgraph A represented by a chromosome
Output: A clique B in A

B ← A
while (B is not a clique) do
    Find all vertices with the smallest and second smallest
         degree in B and store them in an array
    Randomly delete one vertex from the array
    Update the degree of all vertices connected with
        the deleted vertex and other related variables
endwhile
return B
```

**Figure 4:** The clique extraction algorithm

that is randomly chosen from the list of vertices with smallest and second smallest degree. The algorithm is shown in Figure 4.

### 4.6.2 Clique Improvement

After extracting a clique from a newly generated chromosome, the clique improvement algorithm attempts to increase the size of the clique. We randomly select a position in the chromosome. First from this point to the end of the chromosome, then from the beginning of the chromosome to this point, we check each vertex that is not in the clique achieved so far. If it is connected to all vertices in the clique, then we add it into that clique. The algorithm is shown in Figure 5.

```
Input: A clique B represented by
              a chromosome P[0, 1, ..., |V|-1]
Output: A chromosome representing a clique that contains B

Randomly select a position i in chromosome P
for j = i to |V|-1
    if P[j] = 0 then
       if (j is connected to all vertices
                   in the clique represented by P)
       then
          P[j] ← 1
       endif
    endif
endfor
for j = 0 to i
    if P[j] = 0 then
       if (j is connected to all vertices
                   in the clique represented by P)
       then
          P[j] ← 1
       endif
    endif
endfor
return P[0, 1, ..., |V|-1]
```

**Figure 5:** The clique improvement algorithm

### 4.7 Replacement

After applying the local optimization to offspring, the next step is replacement. The replacement step needs to decide which offspring is better and whether the better offspring can replace a member of the population. Attempting to maintain the diversity of the population as much as possible, we choose the offspring with better fitness value and compare it with the fitness of a parent who is more similar to the offspring, where similarity is measured by the Hamming distance. If the offspring's fitness is better, we replace it with the parent. If not, we compare it with the other parent and replace the parent if the offspring is better. If the offspring's fitness is worse than both parents, we compare it with the worst member of the population and replace the worst member if the offspring's fitness is better. If the offspring is worse than both parents and the worst member of the population, we discard it.

### 4.8 Termination Condition

The algorithm stops if the sum of the fitness of all members in the population does not improve for $s$ generations, where $s$ is called the *stagnancy*. The stagnancy is a tradeoff between the quality of the result and the running time. A larger stagnancy yields better results, but the algorithm then takes longer time. From our experimental results, we set the stagnancy to 50, which yields a good balance of solution quality and running time.

## 5 Test Results

In this section we describe the nine classes of graphs available in the DIMACS benchmark test suite for MAXCLIQUE, and then compare our results with the results achieved by two of the best existing algorithms for MAXCLIQUE.

### 5.1 The Test Graphs

The DIMACS (Center for Discrete Mathematics and Theoretical Computer Science) benchmark graphs for MAXCLIQUE are a collection of 9 differ-

ent classes of graphs for evaluating and comparing different algorithms for solving the MAXCLIQUE. This collection consists of random graphs with known maximum clique size as well as graphs obtained from various areas of applications. The C-FAT graphs are based on the fault diagnosis problem [4]. The Hamming and Johnson graphs are from coding theory problems [19] [25]. The Keller graphs arise from the Keller conjecture on tilings using hypercubes [18]. The SAN, SANR, BROCK and P-HAT are different types of random graphs with known maximum clique sizes. The BROCK graphs are random graphs constructed so that they have hidden cliques that are larger than what might be expected in a random graph. The P-HAT graphs are random graphs with large variance in the vertex degree distribution and a larger clique than the usual random graphs [5]. Finally, the MANN graphs are from the vertex covering problem which is closely related to the MAXCLIQUE problem. [7]

## 5.2 Results and Comparison

We tested our algorithm on the DIMACS benchmark graphs for the MAXCLIQUE problem ranging in size from 200 vertices to 3,361 vertices and edges up to 4,619,898. Our algorithm was implemented in C and run on a Pentium III 1.1GHz processor with 256 MB RAM. For each graph, the algorithm was run for 200 trials. Table 1 gives representative sample results consisting of 23 out of 64 graphs given by DIMACS benchmark test suite for MAXCLIQUE. The graphs that we chose to test are basically the test set for Bui and Eppley's algorithm [7]. We report the best and average clique size as well as the average running time. HGAMC found the maximum clique for most graphs (more than 92%). For C-FAT graphs, JOHNSON graphs, HAMMING graphs, SANR graphs and P-HAT graphs, HGAMC can always find the maximum clique. For other graphs, the best solution obtained by HGAMC is very close to the size of maximum clique.

We compared our algorithm with two of the best existing algorithms for MAXCLIQUE, and the results are shown in Table 2. GAMC is a hybrid

genetic algorithm for solving MAXCLIQUE proposed by Bui and Eppley in 1995 [7]. The notable features of this algorithm are the preprocessing and the adaptive fitness function. It is also the algorithm that we modified. The results show that our algorithm performs better overall. In graphs for which GAMC found the maximum clique, our algorithm also found the maximum clique. In graphs for which GAMC didn't find the maximum clique, our algorithm either found the maximum clique or found a larger clique than the largest clique found by GAMC.

Another algorithm, HGA, was designed by Marchiori in 1998 [20]. HGA incorporates a simple genetic algorithm and a naive greedy heuristic algorithm. We compared the best and average clique size achieved by HGA and our HGAMC. Both algorithms found the maximum clique of 18 graphs consisting of all C-FAT, JOHNSON, HAMMING, SANR graphs and most of KELLER, SAN, P-HAT graphs. For Keller6, p-hat-1500-1 and MANN-a27, our algorithm found better solutions than those of HGA, and Keller-6 is considered as a very difficult graph for GA [21]. However, for san1000 and MANN-a45, the results of our algorithm are worse than the results of HGA. Both algorithms run very fast. We couldn't compare the running time of these two algorithms since they were run on different machines for which we could not determine the time conversion factor. Overall, these two algorithms are comparable.

**Table 1:** Results of HGAMC algorithm

| Graph | Node | Edge | Max Clique | HGMCA Best* | Avg. Size* | Avg. CPU Time(s)* |
|---|---|---|---|---|---|---|
| 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 | 55 | 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 |

* the results are from 200 runs for each graph.

**Table 2:** Comparison of HGAMC results with other algorithms

| Graph | HGAMC (Size) | | GAMC (Size) | | HGA (Size) | | Best |
|---|---|---|---|---|---|---|---|
| | Best | Avg. | Best | Avg. † | Best | Avg. | DIMACS |
| c-fat200-1 | 12 | 12 | 12 | – | 12 | 12 | 12 |
| c-fat500-1 | 14 | 14 | 14 | – | 14 | 14 | 14 |
| johnson16-2-4 | 8 | 8 | 8 | – | 8 | 8 | 8 |
| johnson32-2-4 | 16 | 16 | 16 | – | 16 | 16 | 16 |
| keller-4 | 11 | 11 | 11 | – | 11 | 11 | 11 |
| keller-5 | 27 | 26.4 | 18 | – | 27 | 26.3 | 27 |
| keller-6 | 55 | 51.88 | - | – | 53 | 51.4 | 59 |
| hamming10-2 | 512 | 512 | 512 | – | 512 | 512 | 512 |
| hamming8-2 | 128 | 128 | 128 | – | 128 | 128 | 128 |
| san200-0.7-1 | 30 | 29.6 | 30 | – | 30 | 30 | 30 |
| san400-0.5-1 | 13 | 8.6 | 7 | – | 13 | 9.8 | 13 |
| san400-0.9-1 | 100 | 100 | 50 | – | 100 | 100 | 100 |
| sanr200-0.7 | 18 | 18 | 17 | – | 18 | 17.4 | 18 |
| sanr400-0.5 | 13 | 12.9 | 12 | – | 13 | 11.9 | 13 |
| san1000 | 10 | 9.3 | 8 | – | 15 | 10.5 | 15 |
| brock200-1 | 21 | 20.3 | 20 | – | 21 | 18.2 | 21 |
| brock400-1 | 25 | 24.2 | 20 | – | 25 | 23.6 | 27 |
| brock800-1 | 21 | 20.3 | 18 | – | 21 | 19.2 | 23 |
| p-hat300-1 | 8 | 8 | 8 | – | 8 | 8 | 8 |
| p-hat500-1 | 9 | 9 | 9 | – | 9 | 9 | 9 |
| p-hat700-1 | 11 | 10.4 | 8 | – | 11 | 10.3 | 11 |
| p-hat1000-1 | 10 | 9.6 | 8 | – | 10 | 9.9 | 10 |
| p-hat1500-1 | 12 | 11.1 | 10 | – | 11 | 10.4 | 12 |
| MANN-a27 | 126 | 123.5 | 125 | – | 125 | 125 | 126 |
| MANN-a45 | 339 | 336.2 | 337 | – | 342 | 342 | 345 |

†The average clique sizes obtained by GAMC are not available

# 6 Conclusion

In this paper we presented a hybrid genetic algorithm (HGAMC) for solving the maximum clique problem. The main features of HGAMC are a strong local optimizer, graph preprocessing and adaptive genetic operators. The algorithm was tested on DIMACS benchmark graphs and performs very well overall. On more than 92% of 64 graphs, HGAMC found the maximum clique. For other graphs, the results achieved by HGAMC are very close to the maximum clique. For Keller-6, which is considered to be a very hard problem for MAXCLIQUE, HGAMC found a solution better than any other algorithms. Based on our observation, HGAMC performes relatively poor on graphs having hidden cliques such as BROCK graphs. In terms of running time, HGAMC also performs very well. Overall, it provides very good solutions, comparable to the best algorithms that have been proposed so far.

Future work could consider changing the termination condition. If the algorithm reaches stagnancy, then it can relocate part of the population to a different area of the search space. The algorithm can then be applied the algorithm on those members until the algorithm reaches stagnancy again. This step can be repeated a certain number of times. In this way the algorithm has a chance to escape a local optimum and explores more of the search space. However, we need to find a good relocation method.

# References

[1] E. Aarts and J. K. Lenstra (Eds.) "*Local Search in Combinatorial Optimization*," John Wiley and Sons, Chichester, UK, 1997.

[2] E. Balas and C. S. Yu, "Finding a Maximum Clique in an Arbitrary Graph," SIAM J. Comput., 14, 1986, pp. 1054–1068.

[3] 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.

[4] 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.

[5] M. Brockington and J. Culberson, "Camouflaging Independent Sets in Quasi-Random Graphs," Working Paper, Second DIMACS Implementation Challenge, 1993.

[6] A. E. Brouwer, J.B. Shearer, N. J. A. Sloane and W. D. Smith, "A New Table of Constant Weight Codes," IEEE Trans. Inform. Theory, 36, 1990, pp. 1334–1380.

[7] T. N. Bui and P. H. Eppley, "A Hybrid Genetic Algorithm for the Maximum Clique Problem," Proceedings of the 6th International Conference on Genetic Algorithms(ICGA), Pittsburgh, PA, Morgan Kaufmann, 1995, pp. 478–484.

[8] T. N. Bui and B. R. Moon, "Genetic Algorithm and Graph Partitioning," IEEE Transactions on Computers, 45(7), 1996, pp. 841–885.

[9] M. Garey and D. Johnson, "Computers and Intractability - A guide to the Theory of $NP$-Completeness," Freeman, San Francisco, CA, 1979.

[10] D. E. Goldberg, "*Genetic Algorithms in Search, Optimization and Machine Learning*," Addison-Wesley, Boston, MA, 1989.

[11] M. H. Han and D. Jang, "The Use of Maximum Curvature Points for the Recognition of Partially Occluded Objects," Pattern Recognition, 23, 1990, pp. 21–33.

[12] J. Hastad, "Clique is Hard to Approximate within $n^{(1-\epsilon)}$," Proceedings of the 37th Annual Symposium on the Foundations of Computer Science, Burlington, Vermont, 1996, pp. 627–636.

[13] J. Hertz, A. Krogh and R. G. Palmer, "*Introduction to the Theory of Neural Computation*," Assison-Wesley, Redwood City, CA, 1991.

[14] J. H. Holland, "*Adaptation in Natural and Artificial Systems*," University of Michigan Press, Ann Arbor, MI, 1975.

[15] 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).

[16] R. M. Karp, "Reducibility among Combinatorial Problems," Complexity of Computer Computations, pp. 85 - 103. Plenum Press, New York, 1972.

[17] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing," Science, 220, 1983, pp. 671–680.

[18] J. C. Lagarias and P. W. Shor, "Keller's Cube-Tiling Conjecture is False in High Dimensions," Bulletin AMS, 27(2), pp. 279–283.

[19] J. MacWilliams and N. J. A. Sloane, "*The Theory of Error Correcting Codes*," North-Holland, Amsterdam, 1979.

[20] E. Marchiori, "A Simple Heuristic Based Genetic Algorithm for the Maximum Clique Problem," Proceedings of ACM Symposium on Applied Computing, Atlanta, GA, 1998, pp. 366–373.

[21] J. Marconi and J. A. Foster, "A Hard Problem for Genetic Algorithms: Finding Cliques in Keller Graphs," Proceedings of International Conference on Evolutionary Computing, Anchorage, Alaska, 1998, pp. 650–655.

[22] M. Mitchell, "*An Introduction to Genetic Algorithms*," The MIT Press, Cambridge, MA, 1999.

[23] H. Ogawa, "Labeled Point Pattern Matching by Delaunay Triangulation and Maximal Cliques," Pattern Recognition, 19, 1986, pp. 35–40.

[24] P. M. Pardalos and J. Xue. "The Maximum Clique Problem," Journal of Global Optimization, 4, 1994, pp. 301–328.

[25] N. J. A. Sloane, "Unsolved Problems in Graph Theory Arising from the Study of Codes," J. Graph Theory Notes of New York, 18, 1989, pp. 11–20.

[26] G. Yu, O. Goldschmidt and H. Chen, "Clique, Independent Set, and Vertex Cover in Geometric Graphs," ORSA/TIMS Presentation, San Francisco, CA, 1992.

[27] G.Yu, P.Kouvelis and Luo, "A Weighted Vertex Packing Problem for Specially Structured Geometric Graphs Arising in the Design of Electronic Testing Fixtures," ORSA/TIMS Presentation, San Francisco, CA, 1992.