A FAST MAXIMUM CLIQUE ALGORITHM

A Thesis
Presented
to the Faculty of
California State University, Chico

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
©David Del Maschio 2003
Spring 2003
A FAST MAXIMUM CLIQUE ALGORITHM

A Thesis
by
David Del Maschio
Spring 2003

APPROVED BY THE DEAN OF THE SCHOOL OF
GRADUATE, INTERNATIONAL, AND SPONSORED PROGRAMS:

Robert M. Jackson, Ph.D.

APPROVED BY THE GRADUATE ADVISORY COMMITTEE:

Benjoe Juliano, Ph.D., Chair

Ralph Hizer
PUBLICATION RIGHTS

No portion of this thesis may be reprinted or reproduced in any manner unacceptable to the usual copyright restrictions without the written permission of the author.
ACKNOWLEDGEMENTS

Thanks to Dr. Juliano for letting me talk him into advising me during my inquiry of this highly publicized, and difficult topic. Also thanks to Dr. Hilzer for spending his personal time reading my ideas, and Ingo H. de Boer for WinShell 2.1 which allowed me to use \LaTeX{} efficiently.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publication Rights</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>iv</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>List of Symbols</td>
<td>xi</td>
</tr>
<tr>
<td>Abstract</td>
<td>xii</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
</tr>
<tr>
<td>I. Problem Background and Applications</td>
<td>1</td>
</tr>
<tr>
<td>Mathematical Prerequisites</td>
<td>1</td>
</tr>
<tr>
<td>Graph Design</td>
<td>3</td>
</tr>
<tr>
<td>Pattern Matching</td>
<td>3</td>
</tr>
<tr>
<td>Coding Theory Problems</td>
<td>4</td>
</tr>
<tr>
<td>Hamming Graphs</td>
<td>4</td>
</tr>
<tr>
<td>Johnson Graphs</td>
<td>5</td>
</tr>
<tr>
<td>Analysis of Biological and Archeological Data</td>
<td>6</td>
</tr>
<tr>
<td>Keller’s Conjecture</td>
<td>6</td>
</tr>
<tr>
<td>The Keller’s Graphs $\Gamma_n$</td>
<td>7</td>
</tr>
<tr>
<td>Using C-fat Rings for Fault Diagnosis</td>
<td>8</td>
</tr>
<tr>
<td>of Large Multiprocessor Systems</td>
<td></td>
</tr>
<tr>
<td>Printed Circuit Board Testing</td>
<td>9</td>
</tr>
<tr>
<td>Current Approaches to the Maximum Clique Problem</td>
<td>10</td>
</tr>
<tr>
<td>II. Probabilistic Model</td>
<td>11</td>
</tr>
<tr>
<td>Basic Theory</td>
<td>11</td>
</tr>
<tr>
<td>Approximations</td>
<td>13</td>
</tr>
<tr>
<td>A Guessing Algorithm</td>
<td>15</td>
</tr>
<tr>
<td>Speculations of Success</td>
<td>16</td>
</tr>
<tr>
<td>Concluding Remarks</td>
<td>17</td>
</tr>
</tbody>
</table>
CHAPTER | PAGE
--- | ---
III. Combinatorial Model | 19

Basic Counting Tools | 19
Skeleton Algorithm | 20
Going Greedy | 21
Presenting the Skeleton Algorithm | 23
Bounding Heuristic | 24
Enumeration Scheme | 26
Principal Enumeration Ideas | 27

Essential Subproblems and Search Forests | 27
Constructing the Family of Search Forests \( \mathcal{F}(G) \) | 28
An Example | 28
Constructing a Search Tree | 30
Another Example | 32
Search Tree Generating Algorithm | 34

IV. An Exact Algorithm | 36

Predicting Minimum Distances | 37
Simulating \( i \)-sets | 37
Running the Simulation | 39
More Speculations | 42
A Maximum Clique Algorithm | 43
Computational Experiments | 44
Defending the Results | 46
More Computational Results | 47

V. High Density Strategy | 49

An Example | 49
Putting Trees on Slimfast | 50
Penalties for Smaller Search Space | 51
A High Density Algorithm | 53

VI. Low Density Strategy | 55

Definitions | 56
Partitioning Scheme | 57
LIST OF TABLES

TABLE                                                                 PAGE

1. A partial table to estimate the $C(G)$ to $n$ ratio .......................... 18
2. Numbers of sets containing distance $i$ ............................... 38
3. Refining the simulation ........................................... 40
4. Algorithm (4.4) versus $df max.$ ...................................... 45
5. CPU usage comparisons between parallel and sequential programs ... 47
6. Numbers of descendants for high density components ................. 50
7. Using probability to prune ........................................... 52
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. LogOfNCS()</td>
<td>14</td>
</tr>
<tr>
<td>2. $E()$ routine to calculate the expected number of cliques of size $</td>
<td>C</td>
</tr>
<tr>
<td>3. QuessMaxClique() routine</td>
<td>15</td>
</tr>
<tr>
<td>4. A probabilistic maximum clique algorithm</td>
<td>15</td>
</tr>
<tr>
<td>5. QuessComponentSize() routine</td>
<td>17</td>
</tr>
<tr>
<td>6. A sophisticated heuristic due to Brélaz.</td>
<td>21</td>
</tr>
<tr>
<td>7. The second greedy algorithm</td>
<td>22</td>
</tr>
<tr>
<td>8. The skeleton algorithm</td>
<td>24</td>
</tr>
<tr>
<td>10. Example’s graph $G$</td>
<td>29</td>
</tr>
<tr>
<td>11. Sub-problems for the graph in Figure 10.</td>
<td>30</td>
</tr>
<tr>
<td>12. Search trees for the graph in Figure 10.</td>
<td>31</td>
</tr>
<tr>
<td>13. Simplified search forests covering all at most distance three edges.</td>
<td>33</td>
</tr>
<tr>
<td>14. The enumeration step from distance 1 to distance 1.</td>
<td>33</td>
</tr>
<tr>
<td>15. The step from distance 1 to distance 2.</td>
<td>34</td>
</tr>
<tr>
<td>16. Inducing subgraphs from edges of distance $t$</td>
<td>34</td>
</tr>
<tr>
<td>17. A program to construct $\mathcal{F}$</td>
<td>35</td>
</tr>
<tr>
<td>18. Inducing all subgraphs containing a distance</td>
<td>36</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>19. Generating random $i$-sets</td>
<td>39</td>
</tr>
<tr>
<td>20. A sample simulation output</td>
<td>41</td>
</tr>
<tr>
<td>21. Refining the simulation</td>
<td>42</td>
</tr>
<tr>
<td>22. Algorithm used on DIMACS benchmarking graphs</td>
<td>46</td>
</tr>
<tr>
<td>23. Algorithm proposed for high density graphs</td>
<td>53</td>
</tr>
<tr>
<td>24. $InduceAllChildrenHD$ for high density graphs</td>
<td>54</td>
</tr>
<tr>
<td>25. Algorithm proposed for low density graphs</td>
<td>58</td>
</tr>
</tbody>
</table>
# List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\alpha(G))</td>
<td>cardinality of a maximal independent set</td>
</tr>
<tr>
<td>(C)</td>
<td>clique i.e. a complete sub-graph</td>
</tr>
<tr>
<td>(C(G))</td>
<td>clique number i.e. the order of a maximum clique</td>
</tr>
<tr>
<td>(D_r)</td>
<td>decay rate</td>
</tr>
<tr>
<td>(\delta_G(v))</td>
<td>degree of (v \in V(G))</td>
</tr>
<tr>
<td>(\delta_{\text{min}}(G))</td>
<td>minimum vertex degree in graph (G)</td>
</tr>
<tr>
<td>(\delta_{ij}(v))</td>
<td>number of (i)-cliques containing (v \in V(G))</td>
</tr>
<tr>
<td>(\Delta(G))</td>
<td>density of (G)</td>
</tr>
<tr>
<td>(D(P_k))</td>
<td>subset of essential distances present in (k)-th subproblem</td>
</tr>
<tr>
<td>(E(G))</td>
<td>edge set of graph (G)</td>
</tr>
<tr>
<td>(E(X_s))</td>
<td>expected number of complete subgraphs on (s) vertices</td>
</tr>
<tr>
<td>(E(X_s^i))</td>
<td>same as (E(X_s)) but given the number of (i)-cliques</td>
</tr>
<tr>
<td>(E_j(G))</td>
<td>edge set for distance (j)</td>
</tr>
<tr>
<td>(\mathcal{F}, F_j)</td>
<td>family of search forests, forest for distance (j)</td>
</tr>
<tr>
<td>(G(E, V))</td>
<td>graph (G) with vertex set (V), and edge set (E)</td>
</tr>
<tr>
<td>(\bar{G})</td>
<td>complement graph of graph (G)</td>
</tr>
<tr>
<td>(\Gamma_n)</td>
<td>Keller graph with (n) vertices</td>
</tr>
<tr>
<td>(\Gamma(u))</td>
<td>neighbors of vertex (u \in V(G))</td>
</tr>
<tr>
<td>(\overline{\gamma}[d])</td>
<td>global counts of distance (d)</td>
</tr>
<tr>
<td>(H(n, d), J())</td>
<td>Hamming graph of size (n) and distance (d), Johnson graph</td>
</tr>
<tr>
<td>(K^s)</td>
<td>complete graph on (s) vertices</td>
</tr>
<tr>
<td>(L_j(G))</td>
<td>label set for distance (j)</td>
</tr>
<tr>
<td>(MC)</td>
<td>cardinality of a maximum clique</td>
</tr>
<tr>
<td>(\Omega)</td>
<td>set of graphs with specific parameters</td>
</tr>
<tr>
<td>(P(C(G) \leq n))</td>
<td>probability that (G) has (C(G) \leq n)</td>
</tr>
<tr>
<td>(P_i)</td>
<td>(i)-th subproblem</td>
</tr>
<tr>
<td>(UB)</td>
<td>upper bound</td>
</tr>
<tr>
<td>(S(G))</td>
<td>vertex label set of graph (G)</td>
</tr>
<tr>
<td>(S_f)</td>
<td>set of fixed labels of a subproblem</td>
</tr>
<tr>
<td>(S_c)</td>
<td>cutset of a graph</td>
</tr>
<tr>
<td>(T(u, v))</td>
<td>search tree induced from edge ((u, v))</td>
</tr>
<tr>
<td>(V(G))</td>
<td>vertex set of graph (G)</td>
</tr>
<tr>
<td>(</td>
<td>V</td>
</tr>
<tr>
<td>(i(l_u, l_v))</td>
<td>distance between label (l_u) and label (l_v)</td>
</tr>
<tr>
<td>(\overline{i}[d])</td>
<td>count of distance (d)</td>
</tr>
<tr>
<td>(X_s)</td>
<td>probabilistic label for (K^s) based on number of edges</td>
</tr>
<tr>
<td>(X_s^i)</td>
<td>similar to (X_s) but knowing the number of (i)-cliques</td>
</tr>
</tbody>
</table>
ABSTRACT

A FAST MAXIMUM CLIQUE ALGORITHM

by

©David Del Maschio 2003

Master of Science in Computer Science

California State University, Chico

Spring 2003

This thesis introduces a new algorithm for the maximum clique problem. The algorithm is based on speculations about the labels of the vertices of a maximum clique. The new algorithm was tested on a sub-set of DIMACS benchmarking graphs. The run times of the new algorithm are then compared to the run times of algorithms implemented by other researchers on the same benchmarking graphs. The run times indicate that the algorithm is a good competitor, especially for graphs with density of approximately 75 percent. A partial survey of the maximum clique applications is presented along with the new algorithm, and two of its diversification strategies.
CHAPTER I

PROBLEM BACKGROUND
AND APPLICATIONS

Many practical problems can can either (1) be formulated as a maximum clique problem; or (2) have a sub-problem that requires finding a maximum clique. This necessitates the development of algorithms for the maximum clique problem.

The material presented in this chapter provides an overview of existing literature and documentation regarding the maximum clique problem which can be found in various books. Most of the ideas can be located at Center for Discrete Mathematics & Theoretical Computer Science (a.k.a. DIMACS) ftp site, please refer to the end of this chapter for the actual link.

In general, maximum clique applications include project selection, pattern matching, classification theory, artificial intelligence, retrieval of information, coding theory, computer vision, economics, fault tolerance, signal transmission theory, printed circuit board testing, aligning DNA and protein sequences, and of course graph theory and cryptography. Some of these applications can be found in [4, 45, 47–49].

Mathematical Prerequisites

Let \( G = (V, E) \) be an undirected, unweighed graph where \( V(G) = \{v_1, v_2, \ldots, v_n\} \) is the set of vertices in \( G \), and \( E(G) \subseteq V \times V \) is the set of edges in \( V \). Cardinality of a set \( V \) is denoted by \( |V| \).

Graphs may be represented in computer memory as adjacency matrices. The adjacency matrix of \( G \) is denoted by \( A_G = (a_{ij})_{n \times n} \) where \( a_{ij} = 1 \) if \( (i, j) \in E \),
and \( a_{ij} = 0 \) if \( (i, j) \not\in E \). The complement graph of \( G = (V, E) \) is denoted by \( \overline{G} = (V, \overline{E}) \), where \( \overline{E} = \{(i, j) | i, j \in V, i \neq j \text{ and } (i, j) \not\in E\} \). For a subset \( S \subseteq V \) we call \( G(S) = (S, E \cap (S \times S)) \) the subgraph induced by \( S \). It is also possible to define a subgraph induced by a subset of edges, \( F \subseteq E(G) \); recall that \( E(G) \) is the set of edges of graph \( G \).

**Definition (1.1)** A graph \( G = (V, E) \) is complete if and only if \( \forall i, j \in V, (i, j) \in E \).

**Definition (1.2)** A clique \( C \) is a subset of \( V \) such that the induced graph \( G(C) \) is complete.

The maximum clique problem is to find a clique \( C \) of maximum cardinality in graph \( G \).

**Definition (1.3)** \( C(G) \), the clique number of the graph \( G \), is the number of vertices in a maximum complete subgraph of \( G \).

**Definition (1.4)** A vertex cover \( S \) is a subset of \( V \) such that every edge \( (i, j) \in E \) is incident to at least one vertex in \( S \).

The minimum vertex cover problem is to find a vertex cover of minimum cardinality in \( G \).

**Definition (1.5)** An independent set (stable set, or a vertex packing) is a subset of \( V \), whose elements are pairwise non-adjacent.
The maximum independent set problem is to find an independent set of maximum cardinality. It is easy to see that $S$ is a clique in a graph $G = (V, E) \Leftrightarrow V - S$ is a vertex cover in the complement graph $\overline{G} = (V, \overline{E}) \Leftrightarrow S$ is an independent set of $\overline{G}$.

The maximum clique problem, the vertex cover problem and the maximum independent set problem are equivalent. They can also be classified as \textit{NP}-complete or \textit{NP}-hard, depending on how they are stated (as decision problems or optimization problems). In essence, the \textit{NP}-idea means there exists no algorithm that can solve either problem in polynomial time with respect to the size of the problem. For more details about the maximum clique problem, please refer to [24, 25, 29, 34, 35] (Many papers used in this thesis are due to Dr. Panos Pardalos [31]).

Graph Design

There are known methods to design algorithms for generating graphs with specified clique number, $C(G)$. The algorithms are used in the vertex covering problem, for instance. This problem is equivalent to solving the maximum clique problem for $\overline{G}$, check [38, 39]. It follows that, if $G = (V, E)$ is a graph with minimum vertex cover of size $c$, then the complement graph $\overline{G} = (V, \overline{E})$ has maximum clique size $C(\overline{G}) = |V| - c$ (see also [33]).

Pattern Matching

The technique of matching patterns is in principle graph isomorphism problem, which is known to be related to the maximum clique problem via polynomial reduction. The following paragraph hints a method of how the search for a pattern could be converted to the search for a maximum clique (please see [31]).
Given a target picture and an input picture which involve only a set of points (vertices), create a related compatibility graph with vertices corresponding to pairs of points. Two vertices are adjacent if the corresponding pairs of points are mutually consistent. This can depend on a variety of restrictions, including angular relationships as well as the requirement that no point be matched with more than one other. A large clique represents a large number of mutually consistent pairs, and its size can be used as a measure of the corresponding fit.

Coding Theory Problems

This section will show how coding theory problems can be interpreted as maximum clique problems.

One of the objectives in Coding Theory is to find a binary code as large as possible correcting a certain number of errors for a given size (number of bits) of binary code words. The error correcting codes must consist of binary code words such that any two differ in a certain number of bits. A transmission damaged code word is corrected by being replaced with the corresponding code word from the binary code that differs by the minimum number of bits. For more detail on this topic see [3, 26].

Hamming Graphs

**Definition (1.6)** Hamming distance of binary vectors $u = (u_1, u_2, \ldots, u_n)$ and $v = (v_1, v_2, \ldots, v_n)$ is the number of indices $i$ such that $1 \leq i \leq n$ and $u_i \neq v_i$. It is denoted by $\text{dist}(u, v)$. 
A binary code with code words that have a mutual Hamming distance greater or equal to \( d \) can correct \( \left\lfloor \frac{d-1}{2} \right\rfloor \) errors (see [32] for more details). Thus, it is desirable to find a maximum cardinality set of binary vectors (code words) of size \( n \) with Hamming distance \( d \) (denoted by \( C(n, d) \)).

**Definition (1.7)** Hamming graph \( H(n, d) \), of size \( n \) and distance \( d \) is the graph with vertex set of binary vectors of size \( n \) in which two vertices \( u, v \) are adjacent \( \iff \) \( \text{dist}(u, v) \geq d \). \( C(n, d) \) is the size of a maximum clique in \( H(n, d) \).

It can be shown that graph \( H(n, d) \) has \( 2^n \) vertices, \( 2^{n-1} \sum_{i=d}^n \binom{n}{i} \) edges and the degree of each vertex is \( \sum_{i=d}^n \binom{n}{i} \).

**Johnson Graphs**

Another problem arising from Coding Theory is to find the maximum number of binary vectors of size \( n \) that have precisely \( w \) 1’s and the Hamming distance of any two of these vectors is \( d \). This number is denoted by \( C(n, w, d) \). A binary code consisting of vectors of size \( n \), weight \( w \) and distance \( d \), can correct \( w - \frac{d-1}{2} \) errors [32].

**Definition (1.8)** The Johnson graph, \( J(n, w, d) \), with parameters \( n, w \) and \( d \), is the graph with vertex set the binary vectors of size \( n \) and weight \( w \), where two vertices are adjacent if their Hamming distance is at least \( d \).

The size of the weighted code, \( C(n, w, d) \), equals the size of the maximum clique in \( J(n, w, d) \). It is known that the graph \( J(n, w, d) \) has \( \binom{n}{w} \) vertices, \( \frac{1}{2} \sum_{k=\lceil \frac{w}{n} \rceil}^{w} \binom{n}{k} \binom{n-w}{d-k} \) edges and the degree of each vertex is \( \sum_{k=\lceil \frac{w}{n} \rceil}^{w} \binom{w}{k} \binom{n-w}{d-k} \) (see [9] for more detail).
Analysis of Biological and Archeological Data

Natural sciences use trees to express relationships between objects. Trees can represent the division of a species into two separate classes or the division of features of an item. Sometimes it is necessary to deduce a tree structure from the features of the items. The following is a summary from http://mat.gsia.edu/COLOR/general/cereview/node9.html.

One approach to constructing a tree is to define a distance measure between the items. If the distance measure represents lengths of paths in a tree (from root to leaves), then the hypothetical tree is a good estimate for the actual tree.

During this approach, one constructs a graph whose nodes represent partitions of the items. These partitions are created because items within a partition are closer to each other based on some features than the items compared between the partitions. Two nodes of the graph are adjacent if the partitions are from the same tree.

A clique in this graph represents a set of partitions that can be formed into a tree, so a maximum clique integrates as much of the partition data as possible.

Keller’s Conjecture

This section outlines Keller’s conjecture in order to explain instances of Keller’s graphs among the DIMACS benchmarking graphs. The actual definitions are taken from [3] and are general knowledge. Their presence here is merely for illustration.
**Definition (1.9)** A family of hypercubes with disjoint interiors whose union is the Euclidean space $\mathbb{R}^n$ is a tiling.

**Definition (1.10)** A lattice tiling is a tiling for which the centers of the cubes form a lattice.

Minkowski conjectured that in a lattice tiling of $\mathbb{R}^n$ by translates of a unit hypercube, there exist two cubes that share a $(n - 1)$-dimensional face (more detail regarding this can be found in [33]).

Keller suggested that Minkowski’s conjecture holds in the absence of the lattice assumption. Perron [36] proved Keller’s conjecture for $n \leq 6$. Lagarias and Shor [14] proved that Keller’s conjecture fails for $n \geq 10$. It is left to prove whether the conjecture holds for $n = 7, 8, 9$.

The Keller Graphs $\Gamma_n$

The validity of Keller’s conjecture for the unsolved cases may be investigated on graphs with certain properties. The Keller’s graphs are defined as follows:

**Definition (1.11)** $\Gamma_n$ is a graph with vertex set $V_n = \{(d_1, d_2, \ldots, d_n) : d_i \in \{0, 1, 2, 3\}, i = 1, 2, \ldots, n\}$ where two vertices $u = (d_1, d_2, \ldots, d_n)$ and $v = (d'_1, d'_2, \ldots, d'_n)$ in $V_n$ are adjacent $\Leftrightarrow \exists i$, $1 \leq i \leq n : d_i - d'_i \equiv 2 \text{ mod } 4$ and $\exists j \neq i$, $1 \leq j \leq n : d_j \neq d'_j$.

Corrádi and Szabó [30] presented a graph theoretic version of Keller’s conjecture which says that there is a counterexample to Keller’s conjecture $\Leftrightarrow \exists n \in \mathbb{N}^+ \text{ such that } C(\Gamma_n) = 2^n$. $\Gamma_n$ has $4^n$ vertices, $\frac{1}{2}4^n(4^n - 3^n - n)$ edges, and the degree of each node is $4^n - 3^n - n$. $\Gamma_n$ is very dense and has at least $8^n n!$ different maximum
cliques. One can show that the number of maximum cliques of $\Gamma_n$ is less than or equal to $2^n$ (please see [3, 31] for more detail).

**Using C-fat Rings for Fault Diagnosis of Large Multiprocessor Systems**

In studying the reliability of large multiprocessor systems the task is identifying faulty processors in the system. The classical approach to fault diagnosis is known as the PMC model. It was originally proposed by Preparata, Metze and Chien (see [37]).

In the PMC model, each unit can test other units with the assumption that fault-free units always give the correct results while faulty ones are unpredictable. It is also assumed that the number of faulty units is bound from above. Upon completion of all tests, the results are gathered by a monitoring unit which computes the status of all units.

There are two problems with the PMC model: (1) a fault-free unit may not always detect a faulty one; and (2) the upper bound assumption may be false. Further, the PMC model is accurate only if the upper bound does not exceed the number of neighbors of any unit, which may be wrong for large systems with low connectivity. In order to overcome this problem, distributed fault-tolerance was introduced.

The desire for a more realistic approach to fault diagnosis was achieved by simultaneously relaxing all the three assumptions of the PMC model. The main ideas were proposed by Berman and Pelc (please refer to [3, 26] for more detail).

This approach constructs a class of graphs called $\epsilon$-fat rings, and detects faulty units with a graph algorithm. It is also shown that using this approach, the
probability of correct diagnosis of fault processors is at least $1 - n^{-1}$. For illustration, here is a definition of a $c$-fat ring:

**Definition (1.12)** Let $c$ be a given parameter. A $c$-fat ring is the graph $G = (V, E)$ defined as follows. Let $k = \left\lceil \frac{|V|}{c \log |V|} \right\rceil$ and let $W_0, \ldots, W_{k-1}$ be a partition of $V$ such that $c \log |V| \leq |W_i| \leq 1 + \left\lfloor c \log |V| \right\rfloor$ for $i = 0, 1, \ldots, k - 1$. For $u \in W_i$ and $v \in W_j$ we have $(u, v) \in E \iff u \neq v$ and $|i - j| \in \{0, 1, k - 1\}$.

The graph algorithm proposed by Berman and Pelc incorporates the search for a maximum clique. Some DIMACS benchmarking graphs are also represented by $c$-fat rings (for more detail regarding this topic, please refer to [8]).

**Printed Circuit Board Testing**

A printed circuit board tester determines if a portion of a board is working correctly by placing probes on the board. Due to the size of the probes, not every component can be checked in a single pass. The problem of maximizing the number of components checked in one pass can be formulated as a clique problem by representing a component as a node. Two nodes are adjacent if the corresponding two components cannot be checked simultaneously. A clique is a set of components that can be checked in one pass. This is a geometric clique problem, discussed in a paper by C. Maninno (lost reference).
Current Approaches to the
Maximum Clique
Problem

A very common approach to the problem of finding maximum clique is to use some variant of implicit enumeration. There are also polyhedral approaches \[13\], branch and bound approaches, and quadratic programming. All of these techniques may be deployed in solving the maximum clique problem \[3,4\].

There is a lot of literature covering algorithms for this problem. An extensive survey is (was) available at the time of this writing by anonymous ftp from DIMACS at Rutgers University, NY, ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique. This is also the site of the Second Implementation Challenge in 1993, which called for new algorithms for the maximum clique problem \[13-23\].

While a variety of algorithms and heuristics have been proposed for the solution of the maximum clique problem, only a few of the suggested algorithms have been programmed and tested on graphs where the problem is difficult to solve. Extensive computational results are needed to evaluate the average performance of the algorithms, not only on randomly generated graphs but also on problems from a diverse spectrum of applications.
CHAPTER II

PROBABILITY MODEL

This chapter, similarly as the next chapter, describes avenues which were explored without much success. The motivation for this chapter was to consider the possibility of a purely probabilistic algorithm.

Basic Theory

The following is a slightly modified section of [2]: Let \( G(V, E) \) be an undirected, random graph. Let \( n = |V(G)|, M = |E(G)| \), and let the set \( \Omega = \vartheta(n, M) \) be the set of all graphs with \( M \) edges and \( n \) vertices. For the sake of convenience, view the set \( \Omega \) as a probability space where all graphs have equal probability, \( \frac{1}{M!} \). Let \( N = \binom{n}{2} \). Then all graphic invariants occur as random variables on \( \Omega \), so we may speak of their expected value, standard deviation, etc. We will use the following approximation of Stirling’s formula:

\[
n! \approx \frac{n^n}{e^n} \sqrt{2\pi n}.
\]

Lemma (2.1) Let \( K^8 \) denote a complete graph on \( s \) vertices. Let \( X_s \) denote a complete graph on \( s \) vertices whose existence is speculated. The expected number of \( K^8 \) subgraphs contained in a graph \( G \in \Omega = \vartheta(n, M) \) is

\[
E(X_s) = \binom{n}{s} \binom{N - \binom{s}{2}}{M - \binom{s}{2}} M^{-1}.
\]
Proof: \(|\Omega| = {N \choose M}\). In order to calculate the expected number of \(K^s\) subgraphs in \(G \in \Omega\), first compute the number of graphs \(G \in \Omega\) that contain a fixed complete subgraph \(K_0\) of order \(s\). If \(K_0 \subset G\) then \({s \choose 2}\) of the edges of \(G\) are contained in \(K_0\) and the remaining \(M - \frac{s}{2}\) edges have to be chosen from a set of \(N - \frac{s}{2} = \frac{n}{2} - \frac{s}{2}\) edges. Thus,

\[
\left( \frac{n}{s} \right) = \frac{N - \frac{s}{2}}{M - \frac{s}{2}}
\]

of the graphs \(G \in \Omega\) contain \(K_0\). As there are \(\frac{n}{s}\) choices for \(K^s\), the expected number of \(K^s\) subgraphs contained in \(G \in \Omega\) is as claimed. 

Lemma (2.1) is due to \(\mathbb{E}\). Observe that the prediction is only based on the knowledge of number of edges in \(G\). It seems reasonable to argue that the distribution of edges might not be uniform, and that we can encounter clusters of edges forming cliques of higher cardinality. Suppose the information regarding all triangles in \(G\) is available. It may be then possible to make more precise speculation about the cardinality of maximum clique number, \(C(G)\). This could be done using the following theorem, which is a generalization of Lemma (2.1).

**Lemma (2.2)** Given the number of \(i\)-cliques, \(n(K_i) = M\) in \(G\), the expected number of \(K^s\) subgraphs contained in a graph \(G \in \Omega = \mathfrak{U}(n, M)\) is

\[
E(X_s^i) = \left( \frac{n}{s} \right) \left( \frac{N}{M} - \frac{s}{i} \right) \left( \frac{N}{M} \right)^{-1}.
\]
Here, of course, \( N = \binom{n}{i} \). The proof is similar to the proof of Lemma (2.1), with the observation that an edge is a \( K_2 \).

**Approximations**

Assuming that it is necessary to make approximations to \( E(X^s) \) on a 32-bit machine, both of the Lemmas have to be approximated further. In particular, we use Stirling’s approximation to estimate

\[
\binom{n}{s} = \frac{n!}{s!(n-s)!} \approx \sqrt{\frac{n}{s}} \cdot \frac{n^s}{s^s} \cdot \frac{1}{\sqrt{2\pi(n-s)}}.
\]

This approximation yields

\[\log_{10}\left( \binom{n}{s} \right) = \frac{1}{2} \cdot \log_{10}\left( \frac{n}{s} \right) + n \cdot \log_{10}(n) - s \cdot \log_{10}(s) - (n-s) \cdot \log_{10}(n-s) - \frac{1}{2} \cdot \log_{10}(2\pi(n-s)).\]

Which in turn leads to

\[\log_{10}(E(X^s)) = \log_{10}\left( \binom{n}{s} \right) + \log_{10}\left( \frac{N - \binom{s}{2}}{M - \binom{s}{2}} \right) - \log_{10}\left( \frac{N}{M} \right).\]

The logarithmic approximation of \( \binom{n}{s} \) is now easy to compute. It is also easy to compute \( E(X^s) \) and to make speculations about \( C(G) \). Algorithm (2.1) in Figure 1 approximates \( \log_{10}\left( \binom{n}{s} \right) \). Figure 2 is the approximation of \( \log_{10}(E(X^s)) \) in Algorithm (2.2).
Algorithm (2.1) Program LogONCS\((n,s)\)
1 if \((n \leq 0)\) or \((s \leq 0)\) then return 0
2 return \[
\frac{1}{2} \cdot \log_{10}(\frac{n}{s}) + \\
\frac{n \cdot \log_{10}(n)}{s} - \\
\frac{(n-s) \cdot \log_{10}(n-s)}{2} \\
\]

Figure 1. LogONCS().

Algorithm (2.2) Program \(E(|V|,|C|,|E|)\)
Let \(logval, castoff, m_1, m_2, N, SS\) be temporary variables
1 \(N = |V|\cdot|E|\)
2 \(SS = \frac{|E|}{|C|} \cdot |V|\)
3 \(logval = \text{LogONCS}(|V|,|C|) + \\
\text{LogONCS}(N - SS,|E| - SS) - \\
\text{LogONCS}(N,|E|)\)
4 if \((logval < 0)\) then return 0
5 else
6 \(castoff = logval\)
7 \(m_1 = 10^{castoff}\)
8 \(m_2 = 10^{logval-castoff}\)
9 return \(m_1 \cdot m_2\)

Figure 2. \(E()\) routine to calculate the expected number of cliques of size \(|C|\).

Using Algorithm (2.3) in Figure 3 it is possible to guess the maximum clique size given the number of vertices, and edges of a graph. This could be difficult to do algebraically, because it would be necessary to solve Lemma (2.1) for \(s\) when \(E(X_s^n) = 1\).
Algorithm (2.3) Program QuestMaxClique(|V|, |E|)
Let $C \subseteq V$ denote the current clique
1 $C = \emptyset$
2 $|C| = |C| + 1$
3 while $(E(|V|, |C|, |E|))$
4    $|C| = |C| + 1$
5 endwhile
6 return $|C|$

Figure 3. QuestMaxClique() routine.

A Guessing Algorithm

Given these simple probabilistic tools, it may be entertaining to attack the maximum clique problem using a trivial randomized algorithm. One of the frequently used methods, which is far more sophisticated than the algorithm presented in this chapter is simulated annealing. Here, let us inquire into the difficulties associated with a super simple randomized algorithm, which would find a maximum clique by pure chance. Algorithm (2.4) in Figure 4 is a candidate for this purpose.

Algorithm (2.4) Program Probabilistic
Let $MC$ be the cardinality of the current maximum clique $C$
Let $k$ be a fixed number of trials
1 $MC = 0$
2 while $(trials < k)$
3    $C = \{v\}, v \in V(G)$ picked at random
4    while $(\exists u \in \Gamma(C), \text{ s.t. } u \text{ picked at random})$
5        $C = C \cup \{u\}$
6    if $(|C| > MC)$ then $MC = |C|$
7 endwhile
8 endwhile

Figure 4. A probabilistic maximum clique algorithm.
Speculations Of Success

The probability of finding $C(G)$ with Algorithm (2.4), denoted by $P(C(G) \leq n)$, in a graph on $n$ vertices is

$$P(C(G) \leq n) = C(G)! \cdot \frac{1}{(n-C(G))!},$$

since $P(C(G) \leq n) = \frac{C(G)}{n} \cdot \frac{C(G)-1}{n-1} \cdot \frac{C(G)-2}{n-2} \cdots \frac{1}{n-C(G)+1} = C(G)! \cdot \frac{1}{(n-C(G))!}$. It is immediately obvious that this approach is hopeless on a general graph due to the magnitude of the denominator. The assumption that $\frac{C(G)}{n} = \frac{1}{2} \Rightarrow C(G)! \cdot \frac{1}{(n-C(G))!} \simeq \frac{C(G)^2!}{n!}$.

Definitely, the possibility of some success or a complete failure depends on the $C(G)$ to $n$ ratio. This could be modeled with the following equation:

$$n^k \cdot P(C(G) \leq n) = P_{\text{success}}.$$ 

Here $n^k$ is the requested run time, such as $O(n^2)$, (so $k = 2$ in that case). Given a graph $G(E, V)$, with $|V(G)| = n$ we guess the maximum clique and supply the desirable runtime $n^k$. Define $c < n$ to be the cardinality of a component’s vertex set leading to some success (a reasonably polynomial run time) in search for a clique of size $C(G)$. In other words, we have to solve

$$c! = \frac{(C(G))^2!}{P_{\text{success}}} \cdot n^k.$$

for $c$. This may be difficult algebraically but can be done easily computationally. Consider the following lines.

$$\log_{10}(c!) = 2 \cdot \log_{10}(C(G)!) - \log_{10}(P_{\text{success}}) + k \cdot \log_{10}(n).$$
After using Stirling’s approximation for $c!$ we get

$$\log_{10}(\sqrt{2\pi c} \cdot \frac{c}{e}) \approx 2 \cdot \log_{10}(\sqrt{2\pi C(G)} \cdot \frac{C(G)}{e^{C(G)}}) - \log_{10}(P_{success}) + k \cdot \log_{10}(n).$$

which leads to

$$\frac{1}{2} \log_{10}(2\pi c) + c \cdot \log_{10}(c) - c \cdot \log_{10}(e) \simeq$$

$$\log_{10}(2\pi C(G)) + 2 \cdot C(G) \cdot \log_{10}(C(G)) - 2 \cdot C(G) \cdot \log_{10}(e) - \log_{10}(P_{success}) + k \cdot \log_{10}(n).$$

and to Algorithm (2.5) in Figure 5 ($e$ is the Euler constant $2.71\ldots$). Algorithm (2.5) was used to generate some experimental data, which is in part presented in Table 1.

\textbf{Algorithm (2.5) Program} \texttt{QuestCompSize}(\textit{V}, |E|, \textit{ProbSucc, Runtime})

Let $C_G, c$ be temporary variables

1. $C_G = \texttt{QuestMaxClique}(\textit{V}, |E|)$
2. $c = 2$
3. \textbf{while}
   \begin{align*}
   &\left(\frac{1}{2} \cdot \log_{10}(2\pi c) + c \cdot \log_{10}(c) - \log_{10}(e)\right) \\
   &< \left(\log_{10}(2\pi C_G) \ldots + \ldots + k \cdot \log_{10}(|V|)\right)
   \end{align*}
   \textbf{endwhile}
4. \textbf{return} $c$

\textbf{Figure 5.} \texttt{QuestComponentSize()} routine.

Concluding Remarks

The data in Table 1 suggested that the ratio $C(G)$ to $n$ has to be on average 0.55 for target run time $O(n^2)$. So graph $G$ would have to have a rather large maximum clique, approximately $|\frac{1}{2}V(G)|$ for this approach to succeed. This
Table 1. A partial table to estimate the \( C(G) \) to \( n \) ratio.

<table>
<thead>
<tr>
<th>nodes</th>
<th>density</th>
<th>guessed MC</th>
<th>comp. size for * ( n^2 )</th>
<th>( n^{100} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.6</td>
<td>19</td>
<td>36</td>
<td>183</td>
</tr>
<tr>
<td>1000</td>
<td>0.611</td>
<td>20</td>
<td>37</td>
<td>184</td>
</tr>
<tr>
<td>1000</td>
<td>0.622</td>
<td>20</td>
<td>37</td>
<td>184</td>
</tr>
<tr>
<td>1000</td>
<td>0.633</td>
<td>21</td>
<td>39</td>
<td>185</td>
</tr>
<tr>
<td>1000</td>
<td>0.644</td>
<td>22</td>
<td>41</td>
<td>186</td>
</tr>
<tr>
<td>1000</td>
<td>0.655</td>
<td>22</td>
<td>41</td>
<td>186</td>
</tr>
<tr>
<td>1000</td>
<td>0.666</td>
<td>23</td>
<td>42</td>
<td>187</td>
</tr>
<tr>
<td>1000</td>
<td>0.677</td>
<td>24</td>
<td>44</td>
<td>189</td>
</tr>
<tr>
<td>1000</td>
<td>0.688</td>
<td>25</td>
<td>46</td>
<td>190</td>
</tr>
<tr>
<td>1000</td>
<td>0.699</td>
<td>26</td>
<td>47</td>
<td>191</td>
</tr>
<tr>
<td>1000</td>
<td>0.71</td>
<td>27</td>
<td>49</td>
<td>192</td>
</tr>
<tr>
<td>1000</td>
<td>0.721</td>
<td>28</td>
<td>51</td>
<td>194</td>
</tr>
</tbody>
</table>

(* \( n^i \) is the estimated run time for a component of that size.)

suggests that a simple, purely probabilistic algorithm, such as Algorithm (2.4) will be fairly unlucky in its trivial form of attack. As will be shown later, an exact algorithm with probabilistic pruning will be a fair competitor in the quest for a maximum clique.
CHAPTER III

COMBINATORIAL MODEL

This chapter discusses a combinatorial model which was also explored. Even though this approach itself did not lead to any significant findings, it is included here for completeness.

The main purpose of this model was to exclude vertices in the search graph which cannot be members of the maximum clique vertex set by some simple, and computationally fast means. This was done along the lines of the following: if the degree of a vertex is \( i \), then it cannot be in the vertex set of an \( (i+2) \)-clique. This simple idea was then somewhat generalized using generalized degree as in Definition (3.2), to count triangles containing a vertex instead of adjacent vertices.

Basic Counting Tools

Consider the following definitions of some simple vertex measures. Considering the probability issues, it seems reasonable that the vertex with the highest degree has the highest probability of being contained in a maximum clique. This observation is the basis of several greedy algorithms, such as Algorithm (3.2). Here are some obvious results and definitions.

**Definition (3.1)** Let \( G \) be an undirected graph. Then \( \delta_G(v), v \in V(G) \) denotes the number of edges adjacent to vertex \( v \) the graph \( G \), and is called the degree of the node \( v \) in graph \( G \).

**Definition (3.2)** Let \( G \) be an undirected graph, then \( \delta^i_G(v), v \in V(G) \) denotes the number of \( i \)-cliques containing vertex \( v \) and is called the generalized degree of \( v \).
Lemma (3.1) Let $K_i$ be a complete graph on $i$ vertices, then $\forall v \in V(K_i)$, $\delta_G(v) \geq \binom{i-1}{j-1}$, where $V(K_i) \subseteq V(K_i)$.

Proof: Choose a vertex of $K_i$, this leaves $i - 1$ vertices to choose the remaining $j - 1$ vertices from. ■

Lemma (3.2) Let $K_i$ be a complete graph on $i$ vertices, with $V(K_j) \subseteq V(K_i)$ then $\exists \binom{i}{j}$ of $K_j$’s in $K_i$.

Lemma (3.3) Let $K_i$ be a complete graph on $i$ vertices. Let $i, j, a \in \mathbb{N}$ and $j \leq a \leq i$. Then each complete subgraph of $K_i$ on $j$ vertices, is contained in $\binom{i-j}{a-j}$ complete subgraphs of $K_i$ on $a$ vertices.

Example (3.1) Suppose we have a 4-clique, then we must have $\binom{4}{2} = 6$ complete graphs on 2 vertices (i.e. edges), contained in $\binom{4-2}{3-2} = 2$ triangles each.

An algorithm counting the number of triangles containing a vertex was developed and compared to Algorithm (3.2). The surprising fact was that the apparently more constrained requirement of counting triangles was giving similar results as Algorithm (3.2). At this point this direction of investigation was abandoned.

Skeleton Algorithm

In this section, the skeleton algorithm will be presented. The algorithm uses a standard greedy algorithm for finding a coloring of a graph to estimate the upper bound on the maximum clique in a subgraph. A simple label based enumeration scheme will be introduced and in later chapters developed into a new pruning scheme.
The results on some DIMACS benchmarking graphs will be presented, and based on their results, two new diversification strategies for the maximum clique problem will be outlined.

Going Greedy

The first greedy heuristic is a minimum vertex coloring algorithm due to Brélaz [10] (Algorithm (3.1) in Figure 6), the second algorithm (Algorithm (3.2) in Figure 7) was also used in testing. The two heuristics may be used to establish a lower bound on $C(G)$. Consider the following definition of density of a graph $G$:

**Definition (3.3)** Let $G$ be an undirected graph on $|V(G)|$ vertices and $|E(G)|$ edges, then $\Delta(G)$ denotes density, and

$$\Delta(G) = \frac{|E(G)|}{\binom{|V(G)|}{2}}.$$

**Algorithm (3.1)** **Program**: Dsatur

1. Arrange the vertices by decreasing order of degrees.
2. Color a vertex of maximal degree with color 1.
3. Choose a vertex with a maximal saturation degree.
   If there is an equality, choose any vertex of maximal degree in the uncolored subgraph.
4. Color the chosen vertex with the least possible (lowest numbered) color.
5. If all the vertices are colored, stop. Otherwise, return to 3

**Figure 6.** A sophisticated heuristic due to Brélaz.

The fact that Algorithm (3.2) finds a clique is intuitive, and is also observed algebraically in Lemma (3.4)
Algorithm (3.2) Program Greedy1
1 while($\Delta(G) < 1$)
2 \hspace{1em} if ($\delta(v) = \delta_{\min}(G)$) then $V(G) = V(G) - \{v\}$
3 \hspace{1em} endif
4 \hspace{1em} for $i = 0$ to $|V(G)|$
5 \hspace{2em} if ($u_i \in \Gamma(v)$) then $\delta(u_i) = \delta(u_i) - 1$
6 \hspace{2em} endif
7 \hspace{1em} endfor
8 endwhile

Figure 7. The second greedy algorithm.

Lemma (3.4) Let $G$ and $\bar{G}$ be a graph on $n$ and $n-1$ vertices respectively, where $\bar{G}$ is obtained from $G$ by removing vertex $v$ of $G$ with $\delta_{\min}(v)$. Also let $G$ not be a regular, empty or a complete graph. Then

$$\Delta(G) < \Delta(\bar{G}).$$

Proof: Let $n = |V(G)|$ and $e = |E(G)|$, then $2e$ are all the edges of $G$ counted twice. Since $\delta_{\min}(v)$ is the minimal vertex degree in $G$, we must have that $n\delta_{\min}(v) < 2e$. As a consequence we get that $-n\delta_{\min}(v) > -2e \Rightarrow ne - n\delta_{\min}(v) > ne - 2e \Rightarrow n(e - \delta_{\min}(v)) > e(n - 2) \Rightarrow \frac{e - \delta_{\min}(v)}{n(n-1)} > \frac{e}{n(n-1)}$. 

Observe that for a complete graph $\Delta(G) = 1$. Therefore it makes sense to repeatedly apply Lemma (3.4) and converge to a clique of $G$.

Greedy algorithms are known for not always being able to find the best solution. It is possible to force conditions when this failure can be avoided. One of the possibilities is to decompose the main problem into smaller sub-problems such that at least one is expected to contain a maximum clique. Let $P_k$ denote the $k$-th sub-problem, then the data in Table 1 suggest, that there is some chance in
finding a maximum clique in a sub-problem $P_k$, if the $C(G)/|V(P_k)|$ ratio is about 0.55. Even Algorithm (2.4) could be deployed once the main problem is decomposed into sub-problems such as $P_k$ with $|V(P_k)| \leq \frac{C(G)}{0.55}$. During experimentation with Algorithm (3.1) and Algorithm (3.2) it appeared that the maximum clique to $|V(P_k)|$ ratio is about 0.5 when a maximum clique was found in $P_k$.

Presenting the Skeleton Algorithm

In this section, an exact algorithm for the maximum clique is presented. The purpose is to (1) model the main structure of the algorithm, which will be used in subsequent algorithms; and (2) to show how to induce a sub-problem from an edge, using a greedy algorithm to update the solution. Both of these tasks are incorporated into Algorithm (3.3) in Figure 8. This exact algorithm has several impediments which will be partially fixed later. Two of the most obvious fixes are multiple regenerations of a solution and no pruning facility.

Note that given two vertices $u, v \in V(G)$ such that $(u, v) \in E(G)$, the subgraph induced by vertex set $K = \{\Gamma(u) \cap \Gamma(v)\} \cup \{u, v\}$ contains less vertices than $G$, unless $G$ is a complete graph. The cardinality of $K$ depends on the structure of the original graph $G$, but it is possible to estimate the cardinality of $K$, using the assumption that the density in the parent graph is reasonably uniform. Consider the fact that a vertex of graph $G$, with density $\Delta(G) = z$ has roughly $z \cdot n$ neighbors. Therefore the probability that a vertex has any particular neighbor is $z$, and the probability that both vertices on the inducing edge have a particular common neighbor is $z^2$. Based on these ideas, an estimate of the average cardinality of the vertex set of $i$-th descendant graph can be made. One of the approaches is as follows:
Definition (3.4) Let $P$ be a simple graph, and $\Delta(P) = z$. Then the decay rate $D_i$ is the average rate of decrease in the number of vertices of $i$-th descendant of graph $P$, $i > 0$. It can be calculated as $D_i(P_i) = z^{2^i}$.

Using Definition (3.4) the number of vertices in the $i$-th descendant of a graph $P$ can be approximated as $|V(P_i)| \simeq D_i(P_i) \cdot |V(P)|$.

Algorithm (3.3) Program

1. Let $G$ be the graph being searched for a max clique
2. Let $P_1$ be a sub-graph of $G$
3. Let $P_{i+1}$ be a first descendant of $P_i$
4. Let $MC$ be the current size of the maximum clique
5. Let $GreedyMC[]$ be a greedy algorithm such as Algorithm (3.1)
6. Let $L$ be a family of subgraphs of $G$

1. $L = \emptyset$
2. $L = L \cup \{G\}$
3. $MC = 0$
4. while ($L \neq \emptyset$)
5.      get a graph, say $P_i$, from $L$
6.      if ($GreedyMC(P_i) > MC$)
7.         $MC = GreedyMC(P_i)$
8.      if ($|V(P_i)| > MC$)
9.         for all edges $(u, v) \in E(P_i)$
10.            induce a child graph $V(P_{i+1}) = \Gamma(u) \cap \Gamma(v)$
11.            $L = L \cup \{P_{i+1}\}$
12. endfor
13. endwhile

Figure 8. The skeleton algorithm

Bounding Heuristic

This section presents the bounding procedure published in a paper by C. Mannino and A. Sassano, please refer to [11] for more detail.
A clique covering is a set of cliques of $G = (V, E)$ such that each node of $G$ is contained in exactly one clique. A generalized clique covering is a set of cliques where each node is contained in one or more cliques. Let $c_i$ denote the number of cliques containing a node $v_i \in V$, and let $\bar{c}$ denote the smallest $c_i$, for all $i$. Then one can prove Lemma (3.5).

**Lemma (3.5)** Let $G = (V, E)$ be an undirected unweighed graph containing no double edges and self loops, and $C = \{K_1, K_2, K_3, \ldots K_q\}$ be a generalized clique covering of $G$. Then $|C|/\bar{c} \geq \alpha(G)$.

*Proof:* Please see [11].

A generalized covering may be used to yield a bounding procedure called HEUTHETA (Algorithm (3.4)), also due to C. Mannino and A. Sassano. For more detail in regard to reasoning and design of the bounding procedure, please refer to [11].

The first step of Algorithm (3.4) in Figure 9 is also the most important one. In order to get a clique cover of the smallest cardinality, we have to use the most efficient algorithm available in order to initially obtain a minimal exact cover. For this purpose Algorithm (3.1) was used.

It is now possible to equip Algorithm (3.3) with a simple bounding mechanism. Algorithm (3.4) will be used to compute the upper bounds ($UB$) on the maximum cliques in all descendant graphs. If $UB \leq MC$ for a descendant, the descendant will be pruned.
Algorithm (3.4) Program HEUTHETA
1. Generate a clique covering $C^1$
2. Set the upper bound of $MC$, $UB = |C^1|$ 
3. for $t = 2$ to $m$
4. Set $C_t^j = C_t^{j-1}$
5. for $j = 1$ to $|V|$
6. Select node $v_j$
7. if ($\exists K \in C_{j-1}^j$ such that $K \cup \{v_j\}$ is a clique of $G$)
8. then set $K = K \cup \{v_j\}$ and $C_t^j = C_t^{j-1} \cup \{K\}$
9. else set $K = \{v_j\}$ and $C_t^j = C_t^{j-1} \cup \{K\}$
endfor
10. if ($UB > ||C^t||/t$) then $UB = ||C^t||/t$
endfor

Figure 9. HEUTHETA from [11]

Enumeration Scheme

In this section the structure of the enumeration scheme and some basic procedures for reducing the size of the search tree are discussed. Exploiting of the vertex labels is a simple idea which will help Algorithm (3.3) in reducing the size of the problem.

If it is possible to detect if an edge belongs to a maximum clique, it would benefit line 8 of Algorithm (3.3). Then the first descendant of $G$ induced from this edge contains the maximum clique. The decay rate may be used to estimate the number of edges which have to be selected in order to determine the maximum clique.

Suppose the vertices of $V(G)$ are labeled with non-negative consecutive integers starting at 0. Define a label set of graph $G$ as $S(G) = \{l_u : u \in V(G)\}$, and $l_u \in \mathbb{N}, l_u = 0, 1, 2, \ldots, |V(G)|-1$, and require the mapping $\phi : V \rightarrow S$ be one-to-one and onto. Also define distance between two labels (Definition (3.5)).

**Definition (3.5)** Let distance $l$ between two vertex labels $l_i, l_j$, $l(l_i, l_j) = \min(l_i - l_j, 0)$.
Consider the minimum distance in \( S \), it is obviously 1. Now consider the minimum distance in an \( i \)-subset, say \( M \), such that \( M \in 2^S \) and \( |M| = i \), of the label set of \( S \). It is impossible to know exactly what the local minimum distance in \( M \) is but an upper and lower bound on it is easily found. It is trivial that \( 1 \leq i(l_u, l_v) \leq [\frac{|V|}{i}] \), \( u, v \in V(G) \) and \( l_u, l_v \in M \). The edges induced by the labels within the bounds will be from now on referred to as the essential edges.

Assume that a greedy algorithm found a clique of cardinality \( i \). In order to locate an edge contained in the maximum clique, only the essential edges with the distance within the bounds need to be collected. Each collected edge \((u, v)\) of a specified distance implies the assumption that the vertices \( u, v \in C \).

Principal Enumeration Ideas

The enumeration scheme presented here fixes nodes in pairs (edges) in an organized manner, based on the labels of the nodes. It is necessary to describe several concepts, some of which are already familiar to a computer scientist. The discussion here merely ushers the intended use of the terminology in the following sections.

Essential Sub-problems and Search Forests

**Definition (3.6)** Let \((u, v) \in E(G)\) an essential edge. Then an essential subproblem is a subgraph \( P(V_{u,v}, E_{u,v}) \) of graph \( G(V, E) \) induced by the vertices of \( V_{u,v}(G) = \{\Gamma(u) \cap \Gamma(v)\} \cup \{u, v\} \).

Consider the fact that the edge set \( E_j(G) = \{(u, v) \in E(G) : i(l_u, l_v) = j\} \) contains all edges in \( G \) representing distance \( j \). In that sense, all the essential edges of \( G \) may be enumerated as \( E_{\text{essential}}(G) = E_1(G) \cup E_2(G) \cup \ldots \cup E_j(G) \cup \ldots \cup \)
It also follows that this union is disjoint and that each edge in \( E_j(G) \) has to be used in the construction of a complete enumeration scheme. Observe that each edge set \( E_j(G) \) corresponding to a distance \( j \) can be related to exactly one ordered set \( L_j(G) \) of ordered pairs of labels. The definition of \( L_j(G) \) for essential sub-problems is provided next.

**Definition (3.7)** The set \( L_j(G) = \{(l_u, l_v) \in S(G) \times S(G) : l_v = l_u + j, (u, v) \in E(G)\} \), with the requirement that the ordered pairs are sorted on the first index in ascending order, is called the label set of distance \( j \) corresponding to set \( E_j(G) \).

**Constructing the Family of Search Forests \( \mathcal{F}(G) \)**

It is now easier to classify the essential sub-problems into classes, where each class represents a minimum distance \( j, j = 1, 2, \ldots, |V(G)|/|C| \), presumed to be between two labels in the label set of a maximum clique. In other words, the classes represent a family of search forests \( \mathcal{F}(G) = \{ F_j(G) : j = 1, 2, 3, \ldots, |V(G)|/|C| \} \).

Suppose \( T(u, v) \) denotes the search tree of the essential sub-problem induced from essential edge \((u, v)\). Then one can denote forest \( F_j(G) \in \mathcal{F}(G) \) by \( F_j(G) = \{ T(u, v) : (l_u, l_v) \in L_j(G) \} \), and impose the ordering of \( L_j(G) \) on \( F_j(G) \). Note that \( |F_j(G)| = |L_j(G)| = |E_j(G)| \).

**An Example**

Consider the graph in Figure 10. According to the definitions, \( V(G) = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7\} \), \( E(G) = \{(v_0, v_1), (v_0, v_7), (v_0, v_6), (v_0, v_5), (v_0, v_1), (v_6, v_1), (v_7, v_1), (v_6, v_7), (v_6, v_4), (v_7, v_2), (v_2, v_4), (v_3, v_4), (v_5, v_3), (v_5, v_2)\} \), \( S(G) = \{0, 1, 2, 3, 4, 5, 6, 7\} \).
Figure 10. Example’s graph $G$.

Suppose a greedy algorithm such as Algorithm (3.1) finds triangle $\{v_5, v_6, v_7\}$, then the upper bound on the maximum distance which needs to be collected is $\left\lfloor \frac{8}{3} \right\rfloor = 2$. Since $E_1(G) = \{(v_0, v_1), (v_3, v_4), (v_6, v_7)\}$, $E_2(G) = \{(v_1, v_3), (v_2, v_4), (v_3, v_5), (v_4, v_6)\}$, it follows that $L_1(G) = \{(0,1),(3,4),6,7\}$, $L_2(G) = \{(1,3),(2,4),(3,5),(4,6)\}$. It also follows that $E_{\text{essential}}(G) = E_1(G) \cup E_2(G)$, $F = F_1 \cup F_2$, where search forest $F_1 = \{T(0,1), T(3,4), T(6,7)\}$ and search forest $F_2 = \{T(1,3), T(2,4), T(3,5), T(4,6)\}$. Recall that $T(3,5)$, for instance, is the tree induced from essential edge $(3,5)$. There are seven essential sub-problems, i.e. $P_1 : (0,1), P_2 : (3,4), P_3 : (6,7)$, $P_4 : (1,3), P_5 : (2,4), P_6 : (3,5), P_7 : (4,6)$. For instance, $P_1$ will have search tree $T(0,1)$, set of fixed nodes $S_f = \{0,1\}$, largest label of last distance induced $L_1 = 1$, and last distance induced $l_L = 1$. $P_1$ also has $V(P_1) = \{0,1,6,7\}$ and $E(P_1) = \{(0,1), (0,6), (0,7), (1,6), (1,7), (6,7)\}$. $P_6$ has search tree $T(3,5)$, set of fixed nodes $S_f = \{3,5\}$, largest label of last distance induced $L_1 = 5$, and last distance induced $l_L = 2$. Consult Figure 11 and Figure 12 for sub-problems and search trees.
pertaining to the graph in Figure 10. It turns out that only sub-problem $P_1$ has a first descendant $P_{i+1}$, namely $P_{i+1}$, and this is the only descendant of $P_1$. $P_{i+1}$ has $S_f = \{0, 1, 6, 7\}$ with first fixed edge $(0, 1)$ and edge $(6, 7)$ fixed consequently. $P_{i+1}$ has $L_i = 7$ and $i_L = 1$.

Constructing a Search Tree

The tools to construct the family of search forests $\mathcal{F}$ have been defined, and within the family it is now possible to construct each forest. Within each forest the constituent search trees can be also constructed.

Let $P_k$ be the $k$-th essential sub-problem and let $D(P_k)$ be the sub-set of essential distances (sub-problems) present in $P_k$, then $E(P_k) = \cup_{i \in D(P_k)} E_i(P_k)$. Similar to the previous sub-section, we define $L_j(P_{k+b}), b \geq 0$ for all $b$-th descendants of an essential sub-problem $P_k$.

![Figure 11](image.png)

**Figure 11.** Sub-problems for the graph in Figure 10.
Figure 12. Search trees for the graph in Figure 10.

Definition (3.8) Let \((a,p), \ a, p \in V(G), \ l_0 < l_p\) be the last edge added to the set \(S_f\) of fixed nodes of \(P_k\), let \(l(l_0, l_p) = i, \ b \in N^+ \cup \{0\}\), and \((u, v) \in E(P_{k+b})\). Then the set

\[
L_j(P_{k+b}) = \{(l_u, l_v) \in S(P_{k+b}) \times S(P_{k+b}) : l_v = l_u + j, \ l_u \geq l_p + j\}, \ j = i
\]

\[
\{l_u, l_v) \in S(P_{k+b}) \times S(P_{k+b}) : l_v = l_u + j\}, \ \ j > i
\]

with the pairs ordered on the first index in ascending order, is the ordered set of ordered pairs of labels corresponding to \(E_j(P_{k+b})\) of the \(b\)-th descendant of \(P_k\).

To construct the search tree of essential sub-problem \(P_k\), take each ordered pair \((l, l_m) \in L(P_{k+b})\) from smaller to larger and add the two vertices corresponding to the pair of labels to the fixed set \(S_f\) of \(P_k\), i.e. induce a sub-graph of \(P_{k+b}\) by vertex set \(V(P_{k+b}) \cap \Gamma(l) \cap \Gamma(m)\). Each such addition creates a descendant of \(P_k\). Then reapply Definition(3.8) to each descendant. Note that \(|S_f(P_k)| + 2 \cdot b = |S_f(P_{k+b})|\), and that sub-problem \(P_{k+b}\) has \(\Sigma_{d \in D(P_{k+b})}|L_d(P_{k+b})|\) of first descendants, \(P_{k+b+1}\).

The requirements presented so far suggest how to construct the search tree of an essential edge. Now, define a node in tree \(T(u, v)\), where each node of \(T(u, v)\) represents a sub-problem. This may be done as in Definition (3.9).
Definition (3.9) A $t$-th descendant graph (also called a search node) in search tree $T(u,v)$ is uniquely defined by

- its set of fixed nodes $S_I$
- last distance induced $\ell_L$
- largest label $L_{L_i}$ of last distance induced

Another Example

Purpose of this sub-section is to illustrate Definition (3.8), and to depict the order of edge fixation during enumeration on a hypothetical graph. Figure 13 depicts the order and the combinations of distances to be fixed. Figure 14 depicts the enumeration step from distance 1 to distance 1, and Figure 15 shows the enumeration step from distance 1 to distance 2.
Figure 13. Simplified search forests covering all at most distance three edges.

Figure 14. The enumeration step from distance 1 to distance 1.
Algorithm (3.5) Program InduceAllChildren( sub-problem $P_i$, distance $t$ )

let $P_i = (S_f, l_u, L_i)$ be the $i$-th subproblem
let $u, v \in V(P_i)$, and label of $u$ and $v$ be $l_u, l_v$ respectively.

1. for label $l = |V(P_i)| - t$ to label $l = 0$ set $l = l - 1$
2. if($\exists(u, v) \in E(P_i)$ s.t. $l_u = l, l_v = l + t$ and $u, v \notin S_f(P_i)$ )
3. then
4. endif
5. if(( $l \geq L_i(P_i) + t$) and ( $t = l_u(P_i)$ ) or ( $t > l_u(P_i)$ ) )
6. then
7. induce $P_{i+1}$ from $P_i$, $V(P_{i+1}) = \Gamma(u) \cap \Gamma(v)$
8. in $P_{i+1}$, add $l_u, l_v$ to fixed labels set,
9. $S_f(P_{i+1}) = S_f(P_i) \cup \{u, v\}$
10. in $P_{i+1}$, set last distance induced, $l_{\mu} = t$
11. in $P_{i+1}$, set largest label fixed, $L_{\mu} = l + t$
12. push $P_{i+1}$ on stack
13. endfor

Figure 16. Inducing subgraphs from edges of distance $t$
**Algorithm (3.6)** Program Generate $\mathcal{F}$

- let $G$ be the graph being searched for max clique
- let $S$ be the set of labels of the vertex set
- let $P_i = (S_f, i_L, L_i)$ be the $i$-th subproblem
- let $u, v \in V(P_i)$, and label $u$ and $v$ $l_u, l_v$, respectively.

1. set subproblem iterator $i = 0$
2. set $R_0$ to $G(V, E)$ with $S_f = \emptyset$, $i_L = 1$, $L_i = -1$
3. Use any greedy algorithm to obtain a lower bound on a maximum clique, $MC$
4. push $R_0$ on stack
5. while (stack not empty)
6. set $P_i$ to top of the stack and pop the stack
7. for $t = \lceil |V(G)|/MC \rceil$ to $t = i_L(P_i)$ set $t = t - 1$
   8. Induce All Children($P_i$, $t$) //Algorithm (3.5)
9. endfor
10. endwhile

**Figure 17.** A program to construct $\mathcal{F}$
CHAPTER IV

AN EXACT ALGORITHM

The purpose of this chapter is to introduce an exact algorithm incorporating the results from the previous chapters. This includes bounding and enumeration.

Algorithm (4.1) in Figure 18 is a reasonably fast algorithm; the problem however is, that it most likely will not be much faster than the algorithm presented by C. Mannino and A. Sassano [11]. In order to speed it up a little, some properties of labeled sets will be introduced, which will result in more pruning of $F$.

**Algorithm (4.1) Program** Exact2

- let $G$ be the graph being searched for max clique
- let $UB$ be the upper bound on the maximum clique of $P_i$
- let $P_i = (S_f, t_L, L_t)$ be the $i$-th subproblem
- let $u, v \in V(P_i)$, and label $u$ and $v \ t_u, \ t_v$ respectively.
- let GreedyMC() be a greedy algorithm such as Algorithm(3.1).

```plaintext
1 set sub-problem iterator $i = 0$
2 set $R_i$ to $G(V, E)$ with $S_f = \emptyset$, $t_L = 1$, $L_t = -1$
3 $MC = 0$
4 push $R_i$ on stack
5 while (stack not empty)
6   set $P_i$ to top of the stack and pop the stack
7   $UB = HEUTHETA(P_i)$
8   if ($UB > MC$) then
9     if (GreedyMC($P_i$) > $MC$) then $MC = GreedyMC(P_i)$ endif
10    for $t = \lfloor |V(G)|/MC \rfloor$ to $t = t_L(R_i)$ set $t = t - 1$
11       InduceAllChildren($P_i$, $t$) // Algorithm (3.5)
12     endfor
13   endif
14 endwhile
```

**Figure 18.** Inducing all subgraphs containing a distance
Predicting Minimum Distances

It is clear that in the search for a maximum clique of size $MC$ in a graph with $|V(G)|$ vertices, it would be helpful to identify the edges most likely contained in a maximum clique. It appears that edges mapped into label distances (Definition (3.5)) have a rather distinguishable distribution, which implies that it is possible to sample effectively a large population of the $MC$-sets with only a fraction of $\mathcal{F}$. The simplest way to understand the distribution for use in pruning of $\mathcal{F}$ is through a simulation.

Simulating $i$-sets

Consider the fact that it is possible to conclude whether or not a graph has a clique of size $MC$ by enumerating all $MC$-sets, and by checking if an $MC$-set corresponds to a clique. These subsets of $S(G)$ may be enumerated using Algorithm (3.6).

Suppose it is known how many $MC$-sets contain minimum distance $1, 2, \ldots, \lceil |V(G)|/MC \rceil$. Then it is possible to construct the distribution of counts of $MC$-sets with minimum distance $i$ and speculate about the probability of success in finding a maximum clique in $\mathcal{F}$ reduced to only sub-problems of specified minimum distances which yield desired probability of finding a maximum clique.

The easiest way of approximating the distribution is to run a simple randomized simulation. Consider the results in Table 2. This table was generated by Algorithm (4.2) in Figure 19, and displays the results of guessing the minimum distance for five simulation runs of an experiment where $|V| = 200, MC = 100, 50, 25, 20, 15$.

For example, the row with $MC = 25$, had 9141 sets with minimum distance 1, 849 sets with minimum distance 2, 8 sets with minimum distance 3, and
Table 2. Numbers of sets containing distance \( i \).

<table>
<thead>
<tr>
<th>MC</th>
<th>( \text{minimum sizes contained and counts} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>10000</td>
</tr>
<tr>
<td>50</td>
<td>10000</td>
</tr>
<tr>
<td>25</td>
<td>9141</td>
</tr>
<tr>
<td>20</td>
<td>7790</td>
</tr>
<tr>
<td>15</td>
<td>5589</td>
</tr>
</tbody>
</table>

2 sets with minimum distance 4. Consider how fast the distribution scatters as the \( MC /|V(G)| \) ratio decreases (Table 2 gives some idea about the distribution but it is not intended as a statistical study). Suppose a greedy algorithm finds a \( 25 \) clique in graph \( G \) on 200 vertices. Considering the simulation results, it is possible to speculate that a maximum clique of \( G \) contains two vertices \( u, v \) with labels \( l_u, l_v \), such that \( i(l_u, l_v) = 1 \) with chance \( 9141 / 10000 = 0.9141 \) (with respect to the simulation). Now, suppose that the density of \( G \) is 0.75, and that \( G \) is reasonably uniform. Then there are approximately \( 0.75 \cdot (|V(G)| - 1) = 0.75 \cdot 199 = 150 \) candidate edges \( (u, v) \) such that \( i(l_u, l_v) = 1 \) in \( G \), which need to be used for sub-problem construction. This approach will allow the partitioning of the problem of finding a maximum clique with \( MC \geq 25 \) in a graph on 200 vertices, into approximately 150 sub-problems of finding a maximum clique with \( MC \geq 25 \) in graphs on approximately \( |V(P_i)| \approx D_4(P_i) \cdot |V(G)| = 0.75^2 \cdot 200 = 0.5625 \cdot 200 = 112 \) vertices. These sub-problems contain the maximum clique with chance of approximately 0.9141.

Following the example, it is reasonable to label the vertex sets of all sub-problems with labels \( S(P_k) = \{0, 1, 2, \ldots, 111\} \), re-run the simulation for each one of them, and construct respective search families. This approach appears to be sensible
Algorithm (4.2) Program Distribute Minimum Distances( |K|, |S| )
Let elements of $K \subseteq S(G)$ be chosen at random, and $i \neq j, \forall i, j \in K$
Let $\vec{a}$ be an array of counters, s.t. $\vec{a}[i] = 0, \forall i$
1 for a fixed number of trials
2 generate $K$
3 find the minimum distance between two labels of $K$, $l_{\min}(K)$
4 $\vec{a}[l_{\min}(K)] = \vec{a}[l_{\min}(K)] + 1$
5 endfor

Figure 19. Generating random $\vec{a}$-sets.

due to the fact that each sub-problem graph will have less vertices while the searched
after clique number $C(G)$ will remain the same. This will increase the probability of
locating the edges of maximum clique represented by small distances (such as 1) in
subsequent sub-problems.

Suppose one decides to refine the simulation instead, by inquiring into
counts of a distance $i$ in each tested set. The results in Table 3 were generated by
Algorithm (4.3) in Figure 21, the refined simulation. This algorithm also incorporated
a requirement of Algorithm (4.1) stating that an edge $(u, v)$ is counted if and only
if $u, v \notin S_f$.

It is therefore possible to estimate the chance of a maximum clique having
a count of distance 1 edges, distance 2 edges, and so on. These counts will heavily
prune $\mathcal{F}$ and will assist in obtaining a solution with a specified certificate of optimality.

Running the Simulation

This example shows a way of using the simulation previously discussed
to construct a probabilistic pruning scheme of the search tree $\mathcal{F}$ on DIMACS benchmarking graph, brock200_1 (Figure 20). This graph has 200 vertices and density 0.75.
Suppose a greedy algorithm found a clique of size 19. These values are used as the
### Table 3. Refining the simulation.

<table>
<thead>
<tr>
<th>count of dist. 1</th>
<th>subset size of a 200-set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>7</td>
</tr>
<tr>
<td>22</td>
<td>32</td>
</tr>
<tr>
<td>23</td>
<td>85</td>
</tr>
<tr>
<td>24</td>
<td>199</td>
</tr>
<tr>
<td>25</td>
<td>430</td>
</tr>
<tr>
<td>26</td>
<td>809</td>
</tr>
<tr>
<td>27</td>
<td>1158</td>
</tr>
<tr>
<td>28</td>
<td>1493</td>
</tr>
<tr>
<td>29</td>
<td>1698</td>
</tr>
<tr>
<td>30</td>
<td>1535</td>
</tr>
<tr>
<td>31</td>
<td>1152</td>
</tr>
<tr>
<td>32</td>
<td>749</td>
</tr>
<tr>
<td>33</td>
<td>376</td>
</tr>
<tr>
<td>34</td>
<td>165</td>
</tr>
<tr>
<td>35</td>
<td>74</td>
</tr>
<tr>
<td>36</td>
<td>31</td>
</tr>
<tr>
<td>37</td>
<td>5</td>
</tr>
<tr>
<td>38</td>
<td>0</td>
</tr>
<tr>
<td>39</td>
<td>1</td>
</tr>
</tbody>
</table>
output:
number of vertices: 200 reported max clique: 19 density: 0.75

POPULATION DATA:
7389 sets with min distance 1,
2337 sets with min distance 2,
232 sets with min distance 3,
40 sets with min distance 4,
1 sets with min distance 5,
1 sets with min distance 6.

DISTRIBUTION OF PROBABILITY P OF MIN. DISTANCE i, P(i)
P(1) = 0.7389,
P(2) = 0.2337,
P(3) = 0.0232,
P(4) = 0.004,
P(5) = 0.0001,
P(6) = 0.0001

TO COVER 0.9958 OF POPULATION, ASSUME MIN. DISTANCE AT MOST 3.

2611 sets contained 1 count(s) of distance 1,
4021 sets contained 2 count(s) of distance 1,
2491 sets contained 3 count(s) of distance 1,
745 sets contained 4 count(s) of distance 1,
122 sets contained 5 count(s) of distance 1,
10 sets contained 6 count(s) of distance 1,

1984 sets contained 1 count(s) of distance 2,
3635 sets contained 2 count(s) of distance 2,
2889 sets contained 3 count(s) of distance 2,
1168 sets contained 4 count(s) of distance 2,
274 sets contained 5 count(s) of distance 2,
47 sets contained 6 count(s) of distance 2,
2 sets contained 7 count(s) of distance 2,
1 sets contained 8 count(s) of distance 2,

4626 sets contained 1 count(s) of distance 3,
3796 sets contained 2 count(s) of distance 3,
1322 sets contained 3 count(s) of distance 3,
228 sets contained 4 count(s) of distance 3,
26 sets contained 5 count(s) of distance 3,
2 sets contained 6 count(s) of distance 3

DISTRIBUTION OF SETS CONTAINING i COUNTS OF MIN. DISTANCE
To cover 0.9123 of distance 1 population, collect at most 3 items,
To cover 0.9676 of distance 2 population, collect at most 4 items,
To cover 0.9744 of distance 3 population, collect at most 3 items,
Considering the decay rate you will need to fix at most 3 distances.

Figure 20. A sample simulation output.
simulation input. Figure 20 is the output of an implementation of Algorithm (4.3) which was used to generate the simulation data in Figure 20.

**Algorithm (4.3) Program** \texttt{DistributeDistances(} \( |K|, |S| \) \texttt{)}

Let elements of \( K \subseteq S(G) \) be chosen at random, and \( i \neq j, \forall i, j \in K \)

Let \( \mathbf{a} \) be an array of counters, s.t. \( \mathbf{a}[i] = 0, \forall i \)

Let \( \mathbf{b} \) be a \((2 \times 2)\) array of counters, s.t. \( \mathbf{b}[i][j] = 0, \forall i, j \)

Let \( \mathbf{g}[d] \) be a global array storing the speculated counts of distance \( d \) in a maximum clique.

1. \textbf{for} a fixed number of trials
2. \hspace{1em} generate \( K \)
3. \hspace{1em} find the minimum distance between two labels of \( K, l_{\text{min}}(K) \)
4. \hspace{1em} \( \mathbf{a}[l_{\text{min}}(K)] = \mathbf{a}[l_{\text{min}}(K)] + 1 \)
5. \textbf{endfor}
6. construct distribution of non-zero distances collected in \( \mathbf{a} \)
7. choose a maximal distance which will be collected, based on the results of step 6
8. \textbf{for} a fixed number of trials
9. \hspace{1em} generate \( K \)
10. \hspace{1em} count number of occurrences \( c \) for each collectible distance \( d \) with neither endpoint already used in counting
11. \hspace{1em} \( \mathbf{b}[d][c] = \mathbf{b}[d][c] + 1 \)
12. \textbf{endfor}
13. construct distribution for each distance
14. based on results of step 13, load \( \mathbf{g}[d] \)

**Figure 21.** Refining the simulation.

**More Speculations**

Consider the previous Example. The last sentence of the simulation output says that it is necessary to fix at most 3 edges (6 nodes). Due to the density of this graph, there are \( D_r(P_3) \cdot |V(\text{br\text{o}ck2004\_1})| = 0.0999 \cdot 200 \approx 20 \) nodes in the third descendant. So, the assumption that a greedy algorithm will find a clique bigger than 19 in a graph with about 20 vertices is also made.
The results of the simulation lend themselves to the speculation that if the maximum clique contains minimum distance of one, then about 0.74 of 19-subsets of a 200-set is covered. Suppose also that the maximum clique contains two counts of distance 1, and two counts of distance 2, covering 0.4021 and 0.3635 of the population respectively. The three assumptions would give the chance of success of finding the maximum clique of approximately $0.74 \cdot 0.3635 \cdot 0.4021 = 0.1082$ respective to the simulation.

If we require higher probability of success we could use the simulation results to construct $\mathcal{F}$ reduced to minimum distances 1, 2, and 3. This would give the chance of success $\sim 0.9123 \cdot 0.9676 \cdot 0.9744 = 0.8601$.

Observe that the recommended counts of each distance are respective to the simulation. The counts of each distance to be collected may be adjusted based on run time, and will result in more or less pruning of $\mathcal{F}$.

The experimental code which was used allows for different types of pruning. Specifically, to build $\mathcal{F}$ with all possible combinations and counts of recommended distances, and for a highly constrained sets of distances fixed in an order.

A Maximum Clique Algorithm

Now it is possible to state the final version of Algorithm (4.1). The only difference between the algorithm presented here and Algorithm (4.1) is the incorporation of the recommended counts of each distance, as suggested by simulation in the previous chapter. It is only necessary to keep track of how many and of what distances were used in construction of a subproblem $P_i$. If the count of distance $d$ in a sub-problem $P_i$ is within the limits recommended by the simulation, i.e. count of
distance \( d \) in \( P_i \) is strictly less than \( g_i[d] \), then the distance is collectible in \( P_i \). An extension of Definition (3.9) yields the following definition:

**Definition (4.1)** A \( t \)-th descendant graph \( P_i \) in search tree \( T(u, v) \) is uniquely defined by the following items:

- its set of fixed nodes \( S_f \)
- last distance induced \( l_L \)
- largest label \( L_L \), of last distance induced
- the count of distance \( d \), \( \overline{v}[d] \), collected in \( P_i \) (\( \overline{v}[d] \leq g_i[d], \forall \) essential \( d, t \))

Considering Definition (4.1), it is also necessary to keep track of the recommended counts of each distance. Those are kept in global array \( g_i[] \).

Finally, incorporating the new pruning mechanism into Algorithm (4.1) results in Algorithm (4.4), which is depicted in Figure 22. This algorithm was used on DIMACS benchmarking graphs.

**Computational Experiments**

In this section, the results of two sequential algorithms are compared in Table 4. One of the algorithms (\( dfmax \)) is from the DIMACS ftp site (including reported timings for a Sparc II), and the second is Algorithm (4.4) (times in seconds for a Pentium IV).

Algorithm (4.4) was used with the assumption that the maximum clique contains minimum distance of 1, which gives a reasonable chance of success in most of the cases. Table 4 reports the observed CPU times.
Table 4. Algorithm (4.4) versus $dfmax$

<table>
<thead>
<tr>
<th>name</th>
<th>MC</th>
<th>$P$</th>
<th>algo(4.4)</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>brock200.1</td>
<td>21</td>
<td>381.19</td>
<td>17.776</td>
<td>opt.</td>
</tr>
<tr>
<td>brock200.2</td>
<td>12</td>
<td>0.69</td>
<td>0.22</td>
<td>opt.</td>
</tr>
<tr>
<td>brock200.3</td>
<td>15</td>
<td>4.92</td>
<td>0.65</td>
<td>opt.</td>
</tr>
<tr>
<td>brock200.4</td>
<td>17</td>
<td>21.15</td>
<td>6.169</td>
<td>opt.</td>
</tr>
<tr>
<td>$c - fat200 - 1$</td>
<td>12</td>
<td>0.03</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td>$c - fat200 - 2$</td>
<td>24</td>
<td>0.04</td>
<td>0.24</td>
<td></td>
</tr>
<tr>
<td>$c - fat200 - 5$</td>
<td>58</td>
<td>24238.44</td>
<td>0.34</td>
<td>opt.</td>
</tr>
<tr>
<td>$c - fat500 - 1$</td>
<td>14</td>
<td>0.21</td>
<td>0.811</td>
<td>opt.</td>
</tr>
<tr>
<td>$c - fat500 - 2$</td>
<td>26</td>
<td>0.22</td>
<td>0.972</td>
<td>opt.</td>
</tr>
<tr>
<td>$c - fat500 - 5$</td>
<td>64</td>
<td>75.70</td>
<td>1.67</td>
<td>opt.</td>
</tr>
<tr>
<td>hamming6 - 2</td>
<td>32</td>
<td>0.39</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>hamming6 - 4</td>
<td>4</td>
<td>0.04</td>
<td>0.0000</td>
<td>opt.</td>
</tr>
<tr>
<td>hamming8 - 4</td>
<td>16</td>
<td>44.67</td>
<td>0.18</td>
<td>opt.</td>
</tr>
<tr>
<td>johnson16 - 2 - 4</td>
<td>8</td>
<td>18.60</td>
<td>0.931</td>
<td>opt.</td>
</tr>
<tr>
<td>johnson8 - 2 - 4</td>
<td>4</td>
<td>0.03</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>johnson8 - 4 - 4</td>
<td>14</td>
<td>0.13</td>
<td>0.010</td>
<td>opt.</td>
</tr>
<tr>
<td>keller4</td>
<td>11</td>
<td>9.02</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>MANN.209</td>
<td>16</td>
<td>3.96</td>
<td>0.44</td>
<td>opt.</td>
</tr>
<tr>
<td>p_hat300 - 1</td>
<td>8</td>
<td>0.38</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>p_hat300 - 2</td>
<td>25</td>
<td>18.14</td>
<td>131.93</td>
<td>opt.</td>
</tr>
<tr>
<td>p_hat300 - 3</td>
<td>36</td>
<td>26036.35</td>
<td>256.481</td>
<td>found 34</td>
</tr>
<tr>
<td>p_hat500 - 1</td>
<td>9</td>
<td>1.54</td>
<td>0.902</td>
<td></td>
</tr>
<tr>
<td>p_hat500 - 2</td>
<td>36</td>
<td>4341.54</td>
<td>11.26</td>
<td>found 35</td>
</tr>
<tr>
<td>p_hat700 - 1</td>
<td>11</td>
<td>5.11</td>
<td>2.69</td>
<td>found 10</td>
</tr>
<tr>
<td>p_hat1000 - 1</td>
<td>10</td>
<td>24.44</td>
<td>12.308</td>
<td>opt.</td>
</tr>
<tr>
<td>p_hat1500 - 1</td>
<td>12</td>
<td>234.22</td>
<td>69.32</td>
<td>found 11</td>
</tr>
<tr>
<td>san400.0.5 - 1</td>
<td>13</td>
<td>14660.48</td>
<td>16.36</td>
<td>found 9</td>
</tr>
<tr>
<td>sanr200.0.7</td>
<td>18</td>
<td>76.68</td>
<td>4.77</td>
<td></td>
</tr>
<tr>
<td>sanr400.0.5</td>
<td>13</td>
<td>45.83</td>
<td>10.345</td>
<td></td>
</tr>
</tbody>
</table>

(Time comparisons are made with those at DIMACS.)
Algorithm (4.4) Program HOUND

let $\bar{\nu}$ be the counts of distances already collected in $P_i$
let $\mathbb{G}[d]$ be a global array storing the speculated counts
of distance $d$ in a maximum clique.
let $G$ be the graph being searched for max clique.
let $UB$ be the upper bound on the maximum clique of $P_i$
let $P_i = (S_i, t_u, t_0)$ be the $i$-th subproblem.
let GreedyMC() be a greedy algorithm such as Algorithm(3.1).
let $u, v \in V(P_i)$, and label $u$ and $v$ $t_u, t_v$ respectively.

1. set sub-problem iterator $i = 0$
2. set $R_0$ to $G(V, E)$ with $S_f = \emptyset$, $t_u = 1$, $t_l = -1$
3. $MC = 0$
4. push $R_0$ on stack
5. while(stack not empty)
6. set $P_i$ to top of the stack and pop the stack
7. $UB = HEUHTETA(P_i)$
8. if($UB > MC$) then
9. if(GreedyMC($P_i$) $> MC$) then $MC = GreedyMC(P_i)$ endif
10. for $t = |\bar{\nu}| / d, |MC|$ to $t = l_t(P_i) + 1$ set $t = t - 1$
11. if($\bar{\nu}[t] < \mathbb{G}[t]$) InduceAllChildren($P_i$, $t$)//Algorithm(3.5)
12. endwhile
13. endif
14. endwhile
15. endwhile

**Figure 22.** Algorithm used on DIMACS benchmarking graphs

**Defending the Results**

Observe the failure to find a maximum clique on some graphs, and notice their $MC/|V(G)|$ ratio. The small magnitude of $MC/|V(G)|$ hints a low probability of success with the assumption that the maximum clique contains minimum distance of one. Note that this is typical for graphs of low density.

One possibility of correcting this, is to repeat the algorithm with different assumptions, such as the maximum clique contains minimum distance 2, 3, etc. Another possibility would be to design a special purpose algorithm for low density graphs.
More Computational Results

In this sub-section some results produced by Algorithm (4.4) are presented in Table 5 in comparison with results from DIMACS ftp site obtained by Dr. Panos Pardalos. The times from DIMACS are for an 8-processor 440 Mhz MIPS computer per 1000 runs (for further explanation, please refer to DIMACS). The times obtained for Algorithm (4.4) are for a sequential program run on a PentiumIV processor, and stand for the certificate (a portion of population searched), not for the number of seconds until the clique was found.

**Table 5.** CPU usage comparisons between parallel and sequential programs.

<table>
<thead>
<tr>
<th>name</th>
<th>MC</th>
<th>$P_8$(sec./runs)</th>
<th>algo(4.4)(sec.)</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>san200_0.7_1</td>
<td>30</td>
<td>169.31/111</td>
<td>1.48</td>
<td>opt.</td>
</tr>
<tr>
<td>san200_0.7_2</td>
<td>18</td>
<td>94.09/3</td>
<td>14.121</td>
<td>opt.</td>
</tr>
<tr>
<td>san200_0.9_1</td>
<td>70</td>
<td>75.69/49</td>
<td>67.26</td>
<td>opt.</td>
</tr>
<tr>
<td>san200_0.9_2</td>
<td>60</td>
<td>57.00/8</td>
<td>341.64</td>
<td>opt.</td>
</tr>
<tr>
<td>san200_0.9_3</td>
<td>44</td>
<td>48.35/2 (found 42)</td>
<td>506.00</td>
<td>42</td>
</tr>
<tr>
<td>san400_0.7_1</td>
<td>40</td>
<td>128.22/59</td>
<td>43.645</td>
<td>opt.</td>
</tr>
<tr>
<td>brock400_3</td>
<td>31</td>
<td>654.02/3 (found 25)</td>
<td>268.583</td>
<td>opt.</td>
</tr>
<tr>
<td>brock400_4</td>
<td>27</td>
<td>589.18/72 (found 24)</td>
<td>1354.276</td>
<td>25</td>
</tr>
<tr>
<td>keller5</td>
<td>27</td>
<td>572.95/36</td>
<td>544.604 (certificate ?)</td>
<td>opt.</td>
</tr>
<tr>
<td>hamming10_2</td>
<td>512</td>
<td>943.23/213</td>
<td>200.655 (certificate ?)</td>
<td>opt.</td>
</tr>
</tbody>
</table>

(Time comparisons are made with those at DIMACS FTP site. Here 169.31/111 in $P_8$ column stands for 169.31 seconds per 1000 runs found the optimum 111 times.)

During the experiments conducted on the benchmarking graphs it was noted that the algorithm performed better on graphs from certain sources. For instance, brock-graphs, and C-graphs were relatively easier than Sun-graphs. An interesting speculation is that this may be caused by the distribution of the distances of large cliques in the graphs. The brock-graphs, for example, confirmed to distribution
suggested by the primitive simulation algorithm. It is definitely possible to simulate any distribution, perhaps even distributions not currently known. The sun-graphs appeared to have a rather large number of large cliques with minimum distance of four and five, which is somewhat unusual.

Some possible explanations for this are that distinct classes of graphs have fairly distinct clique distance distributions, or that the designers of the classes of benchmarking graphs unintentionally created unusual vertex labelings. To investigate what the actual reasons for the anomalies are, one could randomly permute the vertex labels of benchmarking graphs and re-apply Algorithm (4.4).

In the next chapters, two new algorithms will be proposed. One will operate on graphs of low density and the other on graphs of very high density. This diversification strategy is due to the limitations observed during deployment of Algorithm (4.4) on those families of graphs.
CHAPTER V

HIGH DENSITY STRATEGY

The algorithm developed in this chapter is a refinement of Algorithm (4.4). The motivation is due to the fact that high density graphs, such as those submitted to the Second DIMACS Implementation Challenge by C. Mannino [22], have practical applications. These graphs have densities in excess of 0.99, and are sometimes encountered in quality control of circuit board testing, for example.

Due to high density, the clique number, \( C(G) \) is expected to be very high, \( \frac{1}{4} \cdot |V(G)| \leq C(G) \leq \frac{1}{2} \cdot |V(G)| \). This implies that a simulation run will recommend to collect only distance 1. Since the decay rate decreases very slowly in the beginning, a large number edges with distance 1 will be collected. This will result in potentially large search trees.

An Example

To illustrate of how large search trees can be for benchmarking graphs, consider the following example for a Mannino graph.

\[
|V(MANN_045)| = 1035, \ C(MANN_045) = 345, \ \Delta(MANN_045) = 0.9963.
\]

Since a greedy heuristic will be used to find a maximum clique in descendant graphs, the cardinality of a descendant’s vertex set has to be on average less than \( 2 \cdot C(G) \) (computational experience) in order for the heuristic to find a maximum clique in the descendant. This simple fact and the high density imply that it is necessary to go down to the \( i \)-th descendant, where according to Definition (3.4) \( i \sim 600 = 0.9963^2 \cdot 1035 \). So \( i \sim 7 \) or 8, which is also the average depth of the search.
tree. Each descendant will produce on average $D_r(P_i) \cdot |V(P_i)|$ sub-graphs. The decay rates may be tabulated for convenience (Table 6).

To obtain a closer estimate, observe that there are going to be approximately $1035 \cdot 0.9963 \approx 1031$ of the first descendants. Then, according to Table 6, each of the first descendants will have 1023 of its own descendants, and so on. All together there will be $1031 \cdot 1023 \cdot 1015 \cdot 1000 \cdot 971 \cdot 914 \cdot 811 \cdot 639 \cdot 396 \approx 1000^8 = 10^{24} \approx 2^{72}$ sub-problems (slightly steep for a PC).

Putting Trees on Slimfast

Large search trees in graphs of high density may somewhat be avoided by considering only sub-sets of edges representing distance one. The edge selection could be done on a probabilistic basis.

Considering the large size of maximum cliques in high density graphs, it is reasonable to expect a high probability of success in a random search of an edge in a maximum clique. Suppose we are willing to pay a penalty in the form of decreased probability of success to take advantage of a slimmer search tree.

**Table 6.** Numbers of descendants for high density components.

| $i$ | $D_r(P_i)$ | $|V(P_i)|$ | number of children |
|-----|------------|------------|--------------------|
| 1   | 0.9926     | 1027       | 1023               |
| 2   | 0.9852     | 1019       | 1015               |
| 3   | 0.9706     | 1004       | 1000               |
| 4   | 0.9421     | 975        | 971                |
| 5   | 0.8875     | 918        | 914                |
| 6   | 0.7877     | 815        | 811                |
| 7   | 0.6205     | 642        | 639                |
| 8   | 0.3850     | 398        | 396                |

(Consider how slowly the decay rate decreases.)
Recall that we are mainly interested in fixing an edge of distance 1 in a descendant, which has not been already fixed in its parent. In order to do this, it is necessary to collect edges until the chance of collecting an edge of a maximum clique reaches a desirable level. It turns out that a geometric random variable can be used to approximate a minimum required number of edges containing an edge of a maximum clique with some probability. (For completeness, a short discussion of the geometric random variable is included in Appendix A).

In order to use a geometric random variable in decreasing the size of the search tree, it is necessary to establish the probability of success of finding a maximum clique edge. Suppose $C(G)$ is the clique number of $G$, then the chance of finding a vertex of a maximum clique at random is $\frac{C(G)}{|V(G)|}$. So the probability of finding an edge is $(C(G)/|V(G)|)^2$. If $C(G)$ is unknown, a greedy estimate may be used.

Once the chance of success is estimated, we proceed as in Algorithm (4.4), with the exception that only a sub-set of the edges in each sub-problem is collected to produce descendants. This collection can be done in at random.

**Penalties for Smaller Search Space**

This sub-section outlines how much the tree size may decrease at the expense of decreased probability of success. Recall that $|V(MANN_{a45})| = 1035$, $C(MANN_{a45}) = 345$, $\delta(MANN_{a45}) = 0.9963$.

Assume that a greedy heuristic found only a 300-clique. This gives the probability of success fully finding a maximum clique edge is $p = (300/1035)^2 \approx 0.084$.

The task is now to solve $\sum_{n=1}^{\infty} p(n) = p\sum_{n=1}^{\infty} (1-p)^{n-1} = \text{desired chance of success}$. This equation has to be solved for $n$ (number of trials).
To obtain an estimate for the number of edges to collect, a simple adding program using the equation above computed the requirement of 58 edges for an expected probability of success 0.91.

Suppose the simulation is run to estimate the distribution of counts of distance 1 in 300-sub-sets of a 1035-set. Suppose the simulation results in a distribution with minimum number of counts 43, peak number of 300-sub-sets having 56 counts of distance 1, and no 300-sub-set having more than 72 counts of distance 1. Then the chance of finding a distance 1 edge for construction of the first descendants containing the maximum clique is approximately \( \frac{43}{1035} = 0.0415 \). The number of trials (number of edges) needed to be collected is approximately 71 for chance of success of 0.91.

As mentioned before the further the descendant the smaller is its vertex set. So because we are searching for a 300-clique, in smaller components, the probability of success increases, which implies that a smaller number of edges needs to be collected in each consequent descendant. These numbers are tabulated in Table 7.

**Table 7.** Using probability to prune.

| \( i \) | \( D_s(P_i) \) | \( |V(P_i)| \) | unrestricted | restricted | \( P(\text{success}) \) |
|------|--------------|---------------|--------------|------------|----------------|
| 0    | 0.9963       | 1035          | 1031         | 67         | 0.9            |
| 1    | 0.9926       | 1027          | 1023         | 67         | 0.9            |
| 2    | 0.9852       | 1019          | 1015         | 66         | 0.9            |
| 3    | 0.9706       | 1004          | 1000         | 64         | 0.9            |
| 4    | 0.9421       | 975           | 971          | 61         | 0.9            |
| 5    | 0.8875       | 918           | 914          | 61         | 0.9            |
| 6    | 0.7877       | 815           | 811          | 56         | 0.9            |
| 7    | 0.6205       | 642           | 639          | 49         | 0.9            |
| 8    | 0.3850       | 398           | 396          | 0          | 1              |

(There are *unrestricted* number of children and *restricted* number of edges to be examined after applying the probabilistic selection criteria.)
The number of sub-problems in the slimmer tree will be approximately
\[ 67 \cdot 67 \cdot 66 \cdot 64 \cdot 61 \cdot 61 \cdot 56 \cdot 49 \sim 7 \cdot 7 \cdot 6 \cdot 6 \cdot 6 \cdot 5 \cdot 10^8 = 10^{14} \sim 2^{42} \]
with chance of success approximately \( 0.9^8 \approx 0.43 \).

A High Density Algorithm

It is now possible to make adjustments to Algorithm (4.4) and Algorithm (3.5) to produce Algorithm (5.1) in Figure 23 and Algorithm (5.2) in Figure 24. The modifications have not been verified experimentally. The suggested changes for graphs of high density only indicate a possible direction for future research.

**Algorithm (5.1) Program HDens**

Let \( G \) be the graph being searched for max clique
Let \( P_i = (S_j, \ell, L_d) \) be the \( i \)-th subproblem
Let \( MC \) be the current size of a maximum clique
Let \( GreedyMV() \) be a greedy algorithm such as Algorithm (3.1)
Let \( UB \) be the upper bound for the maximum clique

1. set \( R_b \) to \( G(V, E) \) with \( S_j = \emptyset \), \( \ell \), \( L_d = 1 \), \( L_d = -1 \)
2. \( MC = 0 \)
3. push \( R_b \) on stack
4. while stack not empty
5. set \( P_i \) to the top of the stack and pop the stack
6. \( UB = HEUTHETA(R_i) \)
7. if \( UB > MC \) then
8. if \( GreedyMC(P_i) > MC \) then \( MC = GreedyMC(P_i) \) endif
9. InduceAllChildrenHD(\( R_i \)) // Algorithm (5.2)
10. endif
11. endwhile

**Figure 23.** Algorithm proposed for high density graphs.
Algorithm (5.2) Program InduceAllChildernHD(Subgraph $P_i$)

Let $u, v \in V(P_i)$, and label of $u$ and $v$ be $L_u, L_v$ respectively
Let $N(i)$ be the required number of edges collected in $i$-th descendant
Let $j$ be the count of edges already collected in a sub-problem
Let $P_i = (S_f, u, L)$ be the $i$-th subproblem

1. $j = 0$
2. for label $l = |V(P_i)| - t$ to label $l = 0$ set $l = l + 1$
3. if ($\exists (u, v) \in E(P_i)$ s.t. $l_u = l, l_v = l + 1$ and $u, v \notin S_f(P_i)$)
4. then
5. if ($l \geq L_i(P_i) + 1$) and ($N(i) > j$)
6. then
7. induce $P_{i+1}$ from $P_i$, $V(P_{i+1}) = \Gamma(u) \cap \Gamma(v)$
8. in $P_{i+1}$, add $L_u, L_v$ to fixed labels set,
   $S_f(P_{i+1}) = S_f(P_i) \cup \{u, v\}$
9. in $P_{i+1}$, set last distance induced, $L_l = 1$
10. in $P_{i+1}$, set largest label fixed, $L_{l} = l + 1$
11. push $P_{i+1}$ on stack
12. $j = j + 1$
13. endif
14. endif
15. endfor

Figure 24. InduceAllChildernHD for high density graphs.
CHAPTER VI

LOW DENSITY STRATEGY

A diversification strategy for low density graphs is presented in this chapter. The reason for this diversification is suggested by the fact that Algorithm (4.4) failed to find an optimum solution on benchmarking graphs of low density. This diversification strategy partitions a graph using a minimum cut algorithm, resulting in three component graphs $G_1$, $G_2$, and cutgraph component, $G_c$. The strategy then recursively partitions the components until the deployed greedy algorithm becomes effective in finding a maximum clique in one of the subsequent components.

Using a minimum cut procedure makes sense in this case due to low density, which should be strictly less than 0.5. It will be shown that if the density is below 0.5, it is possible that the component graphs $G_i$, $i = 1, 2$ can have $|V(G_i)| > 1$. The graph induced by the vertices of the edges in the cutset must also be examined for cliques. It is therefore desirable to induce a cutgraph as small as possible. It is not absolutely necessary to find the actual minimum cut so some random sampling techniques (see [9] for more detail) may be used to obtain a close to optimal cut with a very fast randomized algorithm.

There are several algorithms to compute a minimum cut in a graph. Some of them are the classical network flow-based algorithms as in [9] and [10]. There has been a recent discovery of an edge contraction algorithm implemented originally by K. Melhorn and published in [7]. The most recent and also the fastest one is due to D. Karger [6].
Definitions

**Definition (6.1)** Let $G$ be an undirected graph, then $S_c \subseteq E(G)$ denotes a set of edges removal of which will result in a disconnected or an empty graph. $S_c$ is called a cutset.

**Definition (6.2)** Let $G$ be an undirected graph, then $G_c$ denotes the subgraph of $G$ induced by the vertex set $V(G_c) = \{u, v \in V(G) : (u, v) \in S_c\}$.

Definition (6.1) suggests that if cut-set $S_c$ is removed from $E(G)$, a graph $G$ will be partitioned into at least three components. Observe that there can only be one, not necessarily connected cut-graph $G_c$. A minimal set of components can be labeled as $G_1$, $G_2$ and $G_c$.

**Lemma (6.1)** Let $G$ be an undirected, unweighed graph containing no double edges and self loops. If $\delta_{\text{min}}(G) \geq \lceil \frac{n}{2} \rceil$, then $|S_c| = \delta_{\text{min}}(G)$.

*Proof:* See [3].

It is evident from Lemma (6.1) that if $\delta_{\text{min}}(G) \geq 0.5 \cdot n$, one of the components will be a single vertex, so decomposition will result in component cardinalities $|G_1| = |V(G)| - 1$, $|V(G_2)| = 1$, $|V(G_c)| = \delta_{\text{min}}(G) + 1$. Suppose a minimum cut algorithm will be used only on graphs for which the supposition of Lemma (6.1) is false. Then it makes sense to expect a minimum cut algorithm to partition the graph into components with cardinalities of their vertex sets closer to $\frac{1}{3} \cdot |V(G)|$.

Lemma (6.2) claims that partitioning graph $G$ by removing its cut-set $S_c$, will not destroy any cliques of $G$ even if some of their edges are contained in $S_c$. 
Lemma (6.2) Let $G(E, V)$ be a connected undirected graph and let $S_e$ be a cutset of $G$. Then after partitioning of $G$ by removing its cut-set $S_e$, any clique of $G$ is contained in at least one of $G$'s constituent components.

Proof: Note that it is only possible to destroy a clique by removing at least one edge from it. Suppose the claim is false, then $\exists e \in S_e$ s.t. $e \in$ an unprocessed clique, say $K_i$. $|V(K_i)| \geq 3$ since otherwise the only edge connecting the two vertices has already been processed. So any minimal unprocessed clique is a triangle $T$, with say $V(T) = \{u, v, w\}$. Suppose vertex $w$ was not reached by any cut edges during the phase of decomposition and induction of $G_e$. Therefore $uv$ is a cut edge with $u$ in a different component than $v$. But there is a path $u - w - v$ connecting different components hence $S_e$ is not a cutset. ■

Partitioning Scheme

It is now possible to partition $G$ into components such that $|V(G_i)| > MC$, where $MC$ is a lower bound on $C(G)$. In other words, if a greedy heuristic finds a clique of size $k$, then a component with vertex set of cardinality less then or equal to $k$ is not interesting. It may be possible to obtain an almost recursive bisection of $G$ on every application of a minimum cut algorithm.

The low density strategy is now expressible in the form of Algorithm (6.1) (Figure 25). This algorithm should mainly be used to partition a low density graph into components of higher density. When the density of the individual components approaches 0.5 Algorithm (4.4) would be used.
**Algorithm (6.1)** Program LowDensity

1. let $G$ be the graph being searched for max clique
2. let $L$ be a family of components
3. let $G_i$ be the $i$-th subproblem/component
4. let $UB$ be the upper bound on maximum clique of current components
5. let $MC$ be $|V|$, current maximum clique

1. set $L = \{G\}$
2. set $G_0$ to $G(V, E)$
3. $MC = 0$
4. while ($L \neq \emptyset$ )
5. remove $G_i$ from $L$
6. $UB = HEUHTETA(G_i)$
7. if ($UB > MC$) then
8. use a minimum cut algorithm to decompose $G_i$ into $G_{i1}, G_{i2}$.
9. induce $G_{i,c}$ from the cut set $S_c$
10. if ($GreedyMC(G_{i1}) > MC$) then $MC = GreedyMC(G_{i1})$ endif
11. if ($GreedyMC(G_{i2}) > MC$) then $MC = GreedyMC(G_{i2})$ endif
12. if ($GreedyMC(G_{i,c}) > MC$) then $MC = GreedyMC(G_{i,c})$ endif
13. $L = L \cup \{G_{i1}, G_{i2}, G_{i,c}\}$
14. endif
15. endwhile

**Figure 25.** Algorithm proposed for low density graphs
BIBLIOGRAPHY


APPENDIX
A.1 The Geometric Random Variable

Suppose that independent trials, each with probability \( p \) of success are run until a success occurs. Let \( X \) be the number of trials until the first success, then \( X \) is called a \textit{geometric random variable} with parameter \( p \). Its probability mass function is given by

\[
p(n) = P\{X = n\} = (1 - p)^{n-1} p, \quad n = 1, 2, 3, \ldots
\]

Note that the first \( n-1 \) trials are failures and \( n \)-th trial is a success. For more detail, please refer to [50].

A.2 Alternative Formulations of the Maximum Clique Problem

The maximum clique problem has many equivalent formulations. It may be formulated as an integer programming problem, or as a continuous nonconvex optimization problem. The simplest one is the following \textit{edge formulation}:

\[
\max \sum_{i=1}^{n} w_{ij} x_{ij},
\]

s.t. \( x_{i} + x_{j} \leq 1, \quad \forall (i, j) \in E, \)

\[
x_{i} \in \{0, 1\}, \quad i = 1, \ldots, n.
\]