INDEPENDENT SET PROBLEMS AND ODD-HOLE-PRESERVING GRAPH REDUCTIONS

A Dissertation

by

JEFFREY S. WARREN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2007

Major Subject: Industrial Engineering

INDEPENDENT SET PROBLEMS AND ODD-HOLE-PRESERVING GRAPH REDUCTIONS

A Dissertation

by

Jeffrey S. Warren

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2007

Major Subject: Industrial Engineering

ABSTRACT

Independent Set Problems and Odd-Hole-Preserving Graph Reductions.
(May 2007)
Jeffrey S. Warren, B.S., Abilene Christian University;
M.S., Texas A&M University
Chair of Advisory Committee: Dr. Illya V. Hicks

Methods are described that implement a branch-and-price decomposition approach to solve the maximum weight independent set (MWIS) problem. The approach is first described by Warrier *e*t. al, and herein our contributions to this research are presented. The decomposition calls for the exact solution of the MWIS problem on induced subgraphs of the original graph. The focus of our contribution is the use of chordal graphs as the induced subgraphs in this solution framework.

Three combinatorial branch-and-bound solvers for the MWIS problem are described. All use weighted clique covers to generate upper bounds, and all branch according to the method of Balas and Yu. One extends and speeds up the method of Babel. A second one modifies a method of Balas and Xue to produce clique covers that share structural similarities with those produced by Babel. Each of these improves on its predecessor. A third solver is a hybrid of the other two. It yields the best known results on some graphs.

The related matter of deciding the imperfection or perfection of a graph is also addressed. With the advent of the Strong Perfect Graph Theorem, this problem is reduced to the detection of odd holes and anti-holes or the proof of their absence. Techniques are provided that, for a given graph, find subgraphs in polynomial time that contain odd holes whenever they are present in the given graph. These techniques and some basic structural results on such subgraphs narrow the search for odd holes.

Results are reported for the performance of the three new solvers for the MWIS problem that demonstrate that the third, hybrid solver outperforms its clique-cover-based ancestors and, in some cases, the best current open-source solver. The techniques for narrowing the search for odd holes are shown to provide a polynomial-time reduction in the size of the input required to decide the perfection or imperfection of a graph.

To Amy

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants Nos. DMI-0217265 and DMI-0521209. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER I

INTRODUCTION

A. Independent Set Problems

For a graph $G = (V, E)$, $S \subseteq V$ is an **independent set** if no vertices in S are adjacent. The **maximum independent set** (MIS) **problem** is to find an independent set of largest cardinality in a given graph. The **maximum weight independent set** (MWIS) **problem** is to find an independent set of highest total weight in a given graph according to some weighting of the vertices. Both problems are NP-hard [1].

Exact solutions to these problems are often desired. Problems from coding theory, plane tiling, combinatorial auctions and coloring can be expressed as independent set problems for which one would require or prefer an exact solution [2]. Two examples follow.

In combinatorial auctions, bidders bid for collections of goods instead of only for single goods. A simple type of combinatorial auction is one in which a set G of goods is for sale, each bid b_i is made for a subset of goods $G_i \subseteq G$, and all bids are made secretly and simultaneously. We are interested in knowing which bids should be accepted to produce the greatest possible return for the seller. We do so by modeling the question as a MWIS problem. We form a graph whose vertices are the sets of goods bid on. Two distinct vertices are adjacent if their intersection is nonempty. The weight of each vertex G_i is the bid b_i made for it. A maximum weight independent set of this graph indicates the desired choice of bids.

The journal model is *IEEE Transactions on Automatic Control.*

The minimum coloring problem for a graph $G = (V, E)$ is to find a partition $S_1, \ldots, S_k$ of V with each S_i an independent set and with k minimum. In [3], Mehrotra and Trick describe a column-generation method for solving the minimum coloring problem. The column-generating sub-problem they use is a MWIS problem on G, with weights determined by the dual variables corresponding to the coverage constraints in the master problem. While improving columns could be produced without optimally solving the MWIS sub-problem, an exact sub-problem solution is required for determining the optimality of a best known solution of the master problem.

Many have worked to solve both the MIS and MWIS problems exactly. Techniques include explicit enumeration of maximal independent sets [4], combinatorial branch-and-bound [5, 6, 7, 8, 9, 10, 11, 12, 13], and continuous formulations under branch-and-bound [14, 15]. We refer the reader to [2] for a discussion of these lines of research. Further discussion of previous relevant methods can be found in Chapters II and III.

We will investigate several approaches for obtaining exact solutions. Chapter II discusses our contributions to a branch-and-price method for solving the MWIS problem first presented in [16]. The research represents a significant addition to the literature on continuous methods for the MWIS problem.

Chapter III discusses combinatorial branch-and-bound methods for solving the MWIS problem. The research described there re-evaluates the merits of several established combinatorial methods for the MWIS problem, clarifies their significance, and improves upon them.

B. Hole Detection

For a graph $G = (V, E)$, $H \subseteq V$ is a **hole** if $G[H]$ is a cycle. A hole is **odd** or **even** as its cardinality is odd or even. A graph is **Berge** if both it and its complement contain no odd holes.

Chudnovsky *et al*. proved in [17] that a graph is perfect if and only if it is Berge. Perfect graphs are an important class of graphs, not least because several NP-hard optimization problems are solvable in polynomial time on them. Chudnovsky and Seymour recently proved in [18] that a graph can be shown to be Berge in polynomial time. This result, however, leaves open the question of how difficult it is to find odd holes themselves. The **odd hole detection problem** is to decide if a graph has an odd hole. While deciding Berge-ness is in P, it is not known if the odd hole exclusion problem is in P. In [19] Bienstock showed that the related problem of deciding if a graph or its complement has an odd hole containing a vertex specified *a priori* is NP-complete. No procedure has been developed that will quickly identify odd holes in a graph. In Chapter IV, we will investigate methods for simplifying the detection of odd holes.

C. Research Objectives

The MWIS problem can be solved exactly using a linear-programming-based branch-and-bound algorithm. The linear programs (LPs) at each node in the branch-and-bound tree can be solved using a decomposition approach to column-generation that calls for the repeated solution of the MWIS problem on subgraphs of the original graph (with vertex weightings adjusted from their original values). We will develop a method to partition the vertex set of a graph to produce the subgraphs to be used by such a column-generation routine as implemented in

[16].

We will develop a combinatorial branch-and-bound algorithm to exactly solve the MWIS problem. The algorithm will exploit and extend the best techniques from previous solvers that use clique covers to provide their bounds.

We will investigate the polynomial-time detection of sets $U \subseteq V$ such that $G[U]$ contains an odd hole if G does. We will analyze the structure of such induced subgraphs.

CHAPTER II

A BRANCH-AND-PRICE APPROACH FOR THE MWIS PROBLEM

A. Introduction

In [16], Warrier *et al*. describe a branch-and-price decomposition approach to solving the maximum weight independent set (MWIS) problem. Section B explains the basic branch-and-price formulation. Further sections in this chapter will address our contributions to that approach.

The decomposition in [16] calls for the exact solution of the MWIS problem on induced subgraphs of the original graph. Our contribution centers on the use of chordal graphs as the induced subgraphs in this solution framework. Section C introduces chordal graphs, and Section D discusses the detection of induced subgraphs that are chordal. Section E explains how the MWIS problem can be solved quickly on chordal graphs, motivating their use as branch-and-price subproblems. We discuss in Sections F and G our other contributions to the approach that are not specific to the choice of decomposition method.

B. The Branch-and-Price Formulation

Consider a graph $G = (V, E)$ and a weight function $w : V \to R$, where $V = \{v_1, \ldots, v_n\}$. A base formulation for an integer program (IP) that solves the MWIS problem for G rests on defining an integer decision variable x_i associated with vertex v_i, where $x_i = 1$ if v_i belongs to a chosen independent set, and $x_i = 0$ otherwise. To guarantee that the x_i values correspond to an independent set, we enforce a constraint $x_i + x_j \leq 1$ for every edge $v_iv_j \in E$. The objective function needed to solve the MWIS problem is $\sum_{i=1}^{n} w(v_i)x_i$.

The decomposition approach of [16] calls for partitioning V as $V_1 \cup \cdots \cup V_k$. Each V_i for $i \in \{1, \ldots, k\}$ induces a subgraph in G having edge set E_i. We define $\hat{E} = E \setminus \bigcup_{i=1}^{k} E_i$ and $\hat{V}$ as the set of vertices incident to at least one edge in $\hat{E}$. We define Q as the set of all binary $|V|$-tuples satisfying all edge constraints corresponding to edges in $\bigcup_{i=1}^{k} E_i$. We also define the vector $\mathbf{w}$ with elements $w_i = w(v_i)$. Then an equivalent IP formulation of the MWIS problem is maximizing $\mathbf{w} \cdot \mathbf{x}$ over all $\mathbf{x} \in Q$ such that $x_i + x_j \leq 1$ for all $v_i v_j \in \hat{E}$. Define the matrix A so that these edge constraints for edges in $\hat{E}$ are represented by $A\mathbf{x} \leq \mathbf{1}$.

Consider each $\mathbf{x} \in Q$, the weight vector $\mathbf{w}$, and the matrix A to be partitioned according to the partition of V:

$$\begin{aligned} \mathbf{x} &= \left[\mathbf{x}^1 \cdots \mathbf{x}^k\right] \\ \mathbf{w} &= \left[\mathbf{w}^1 \cdots \mathbf{w}^k\right] \\ A &= \left[A_1 \cdots A_k\right]. \end{aligned}$$

Then our objective function is equivalent to $\sum_{i=1}^{k} \mathbf{w}^i \cdot \mathbf{x}^i$, and our constraints are equivalent to $\sum_{i=1}^{k} A_i \mathbf{x}^i \leq \mathbf{1}$ for all $\mathbf{x}^i \in Q_i$, $i \in \{1, \ldots, k\}$.

The Dantzig-Wolfe decomposition of the linear relaxation of this IP yields a reduced master problem (RMP) that optimizes over the convex hulls of the Q_i:

$$\begin{aligned} \text{Max} \quad & \sum_{i=1}^{k} \mathbf{w}^i \cdot \left(\sum_{j=1}^{|R_i|} \lambda_{ij} \mathbf{x}^{ij} \right) \\ \text{subject to} \quad & \sum_{i=1}^{k} A_i \left(\sum_{j=i}^{|R_i|} \lambda_{ij} \mathbf{x}^{ij} \right) \leq \mathbf{1} && (2.1) \\ & \sum_{j=1}^{|R_i|} \lambda_{ij} = 1 \qquad \forall i \in \{1, \ldots, k\} && (2.2) \\ & \lambda_{ij} \geq 0 \qquad \forall i \in \{1, \ldots, k\}, j \in \{1, \ldots, |R_i|\}, \end{aligned}$$

where $R_i \subseteq Q_i$ is the set of known extreme points of $\mathrm{conv}(Q_i)$, and the $\mathbf{x}^{ij}$ are the indexed members of R_i. Sub-problem i over Q_i is then

$$\begin{aligned} \text{Max} \quad & (\mathbf{w}^i - A_i\boldsymbol{\pi}) \cdot \mathbf{x}^i \\ \text{subject to} \quad & \mathbf{x}^i \in Q_i, \end{aligned}$$

where $\boldsymbol{\pi}$ is a vector of the $\left|\hat{E}\right|$ dual variables corresponding to Constraint 2.1 of the RMP.

The column-generation process works as follows. We begin by assigning to each R_i some known extreme points of each $\mathrm{conv}(Q_i)$ and solving the RMP. The dual variables $\boldsymbol{\pi}$ are passed to all sub-problems, which are then optimized. For each optimal sub-problem solution $\mathbf{x}^i$, we compute $(\mathbf{w}^i - A_i\boldsymbol{\pi}) \cdot \mathbf{x}^i - \rho_i$, where ρ_i is the dual variable corresponding to the ith equality in Constraint 2.2. If this quantity is positive, then $\mathbf{x}^i$ is an improving column and is added to R_i. We then repeat the process of solving RMP and the sub-problems until no sub-problem solutions are improving columns for RMP.

This procedure, of course, solves only the linear relaxation of our IP. If the optimal solution to this linear relaxation has non-integer-valued variables, then we will proceed with a branch-and-bound approach to find the optimal integer solution.

If we hope to apply Dantzig-Wolfe decomposition to the given IP formulation of the MWIS problem, we will need a method for solving each of the sub-problems that the decomposition produces. Our first contribution to this approach is a method to partition V so that each sub-problem is a MWIS problem on a chordal graph. Our second is an efficient method to solve every sub-problem instance, regardless of additional constraints provided by the branch-and-bound process. The following sections describe these contributions.

C. Chordal Graphs

A graph is **chordal** if all its cycles of length at least four have a chord. Equivalently, a graph is chordal if all its induced cycles are triangles. Clearly, every induced subgraph of a chordal graph is chordal.

Given a permutation $v_1, \ldots, v_n$ of a graph's vertices, a **successor** of v_i is any neighbor v_j of v_i such that $j > i$. If v_i has any successors in the permutation, then the successor with the lowest index is called the **first successor** of v_i.

A permutation of a graph's vertices is a **perfect elimination scheme** (PES) if, for every i, the first successor of v_i is adjacent to all other successors of v_i. A graph is chordal if and only if it has a PES (see [20]). For a chordal graph $G = (V, E)$ with PES σ and $W \subseteq V$, W induces a PES for $G[W]$ in σ.

If, in a general graph $G = (V, E)$, $W \subseteq V$ induces a chordal subgraph, then $G[W]$ is called a **chordal induced subgraph** (CIS) of G.

D. Chordal Graphs and the Vertex-Set Partition

To achieve the partition of the vertex set required for the decomposition described above, we will make use of a method to find a vertex-maximal CIS in a given graph. A vertex-maximal CIS for the graph $G = (V, E)$ can be found in $O(|V| + |E|)$ time using the following algorithm of Balas and Yu [5]. Let $L(v)$ be a sequence of labels for each vertex $v \in V$. Assign the empty sequence to C and empty sequences to $L(v)$ for each $v \in V$. Let $U \leftarrow V$ and $i \leftarrow n$. For general i, choose a vertex $w \in U$ for which $L(w)$ is lexicographically largest. Let $U \leftarrow U \setminus \{w\}$. If w can be prepended to C so that the resulting sequence is a PES, then do so and append i to each $L(v)$ for which $v \in U \cap N(w)$. Last, decrement i by 1 and repeat until $i = 0$. C will then be a PES for a vertex-maximal chordal induced subgraph of G.

Our implementation of the algorithm adds a degree-based tie-breaker to the original: if vertices tie for the lexicographically largest label among all unnumbered vertices in U, then those having the highest degree in $G[U]$ are favored. Further ties are broken arbitrarily.

To achieve the desired partition of V, we do the following. Let $U \leftarrow V$ and let T be the vertex set of a vertex-maximal CIS in $G[U]$. Then let $U \leftarrow U \setminus T$ and repeat, retaining the sets T until U is empty. Then each set in our collection will induce a chordal subgraph in G.

E. The MWIS Problem on Chordal Graphs

The motivating advantage of our partitioning method is that the MWIS problem is efficiently solved on chordal graphs. For the chordal graph $G = (V, E)$, a MWIS can be found in $O(|V|+|E|)$ time using the algorithm of Frank in [21]. We are given $v_1, \ldots, v_n$ as a PES for G and w_i as the weight of v_i for all $i \in \{1, \ldots, n\}$. Inspect the vertices in order from v_1 to v_n and add to the set T the first one having positive weight, say v_i. Reduce the weight of each of its successors by w_i, and find the next vertex with positive weight. Continue until all vertices have been inspected. At that point, all vertices in V are either in $V \setminus T$ with non-positive weight or in T with positive weight.

Now, inspect the vertices of T in reverse order of their indices. Take the first and add it to set S. Continue with the other vertices, adding them to S if none of their successors are in S. When every vertex in T has been inspected, S will be a maximum weight independent set of G.

Solving the MWIS problem using the Dantzig-Wolfe decomposition requires the application of branch and bound. At each node of the branch-and-bound tree, we will specify a set of vertices that are excluded from an optimal independent set and a set of vertices that must be included in an optimal independent set. This can be enforced in RMP by adding appropriate equality constraints on the variables corresponding to the included and excluded vertices.

In the sub-problems, we enforce these inclusions and exclusions by modifying the vertex weights used in Frank's algorithm. For a graph $G = (V, E)$ with PES σ, let $I \subseteq V$ be a set of vertices that we must include in the optimal independent set, and let $X \subseteq V$ be a set of vertices that we must exclude from the optimal independent set. We assume that I and X are disjoint and that X contains at least the neighbors of all vertices in I. We can solve the constrained problem by setting the weight of each excluded vertex to zero and the weight of each included vertex to one and then applying Frank's algorithm.

Frank's algorithm never increases the weight of a vertex, and it never adds to T (or subsequently S) a vertex of weight zero. Thus, it never adds a vertex in X to S. Further, the algorithm never reduces the weight of a vertex in I, because none is ever the successor of a vertex in T. Therefore, it adds every vertex of I to T; it later adds each one to S because none ever has a successor in T, let alone S.

F. Improving the RMP Formulation

For a clique K of G, with vertices $v_{i_1}, \ldots, v_{i_t}$, $\sum_{j=1}^{t} x_{i_j} \leq 1$ is a valid constraint for the base IP formulation in Section B. We call such a constraint a **clique constraint**. Using this constraint in our IP formulation obviates the use of all edge constraints corresponding to edges having their ends in K. In RMP, each edge constraint (the individual inequalities of Constraint 2.1) corresponds to an edge having ends in different sets of the partition of V (*i.e.*, an edge belonging to $\hat{E}$). So, if we can identify cliques of G that include vertices from different V_i, then we will be able to replace edge constraints with clique constraints.

We can do this as follows. Begin by assigning to F the set $\hat{E}$. Choose an arbitrary edge $uv \in F$, find a maximal clique in $G[F]$ (the graph with vertex set $\hat{V}$and edge set F) containing u and v, and then extend it to a maximal clique in G. Remove from F every edge induced by that clique. Repeat, collecting all these cliques, until F is empty.

It would suffice to use clique constraints corresponding to the maximal cliques of $G[F]$ that we find; these would guarantee that every edge constraint of RMP is enforced. The clique constraints for the maximal cliques of G that we find are valid, however. They yield a tighter RMP polytope at little cost compared to the cost of the clique constraints corresponding to maximal cliques of $G[F]$.

G. Initial Feasible LP Solutions

To begin solving the column-generation formulation of the MWIS problem described in Section B, we must initialize the sets R_i with some extreme points of each conv(Q_i). To do so, we use a heuristic for the MIS problem and partition the resulting independent set according to the partition of V.

For every $v \in V$, we find a maximal independent set of G containing v. This results in some $m \leq n$ independent sets $S_1, \ldots, S_m$; depending on the structure of G and the behavior of the heuristic, not all independent sets found will necessarily be unique. Then we compare each independent set to the partition of V: if $V_i \cap S_j \neq \emptyset$ for $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, m\}$, we add a vector corresponding to $V_i \cap S_j$ to R_i.

H. Computational Results

Our partitioning method was tested against another method in [16]. The other method begins with a pre-specified number of parts k and applies the METIS clustering algorithm [22, 23, 24] to produce k parts for V having cardinalities of either $\left\lfloor \frac{V}{k} \right\rfloor$ or $\left\lceil \frac{V}{k} \right\rceil$ and such that the number of edges having ends in two parts of the partition is small. Sub-problems on the resulting subgraphs were solved by a weighted analog of the algorithm of Carraghan and Pardalos [9].

Table I reports results for the two methods on graphs from the Second DIMACS Implementation Challenge [25]; the times are in seconds, and an asterisk (*) denotes instances in which computer memory was exhausted. These results are as reported in [16].

The METIS-based method dominates ours. One reason for this is that, for almost all graphs, METIS leaves a much smaller portion of the graph's edges in $\hat{E}$. So, while our approach may have sub-problems that solve quickly, those sub-problems do not assume much of the computational cost of solving the overall problem.

Another reason is that the two approaches produce sub-problems having different polyhedral structure. If each part V_i induces a chordal graph, then the convex hull of each Q_i is an integer polytope. In [26], Geoffrion demonstrated that such a decomposition, the optimal objective value of RMP is equal to the optimal objective value of the linear relaxation of the original problem. The Q_i produced by METIS do not generally have integer convex hulls. The RMP of a decomposition based on a METIS partition can thus have an optimal objective value less than that of the original linear relaxation. As a result, the METIS-based method will fathom more branch-and-bound nodes and produce a smaller tree.

Table I. The algorithm of Warrier *et al.* on DIMACS graphs

				Chordal			METIS		
Graph	$\|V\|$	$\|E\|$	$\alpha(G)$	Time	Nodes	$\left\|\hat{E}\right\|$	Time	Nodes	$\left\|\hat{E}\right\|$
brock200_3	200	7,852	15	*	>9,000	7,325	2,537.4	3,624	3,463
c-fat200-1	200	18,366	12	158.5	90	16,696	8.9	6	8,999
c-fat200-2	200	16,575	24	261.1	25	15,912	19.2	8	11,406
c-fat200-5	200	8,473	58	*	>13	11,201	86.3	33	9,523
hamming8-2	256	1,024	128	0.6	1	846	0.6	1	626
hamming8-4	256	11,776	16	*	>3,500	10,650	6.1	1	6,528
johnson8-2-4	28	168	4	1.1	8	113	0.2	8	137
johnson8-4-4	70	560	14	11.5	79	433	0.6	1	240
johnson16-2-4	120	1,680	8	1.7	16	1,243	0.6	1	1,082
keller4	171	5,100	11	*	>25,000	4,499	1,812.2	12,523	3,293
p_hat300-1	300	33,917	8	*	>2,000	31,511	479.4	1,086	16,580

I. Conclusions

We have explained in this chapter how a graph's vertex set can be partitioned so that each set in the partition induces a chordal subgraph of the given graph. The MWIS problem on each of these subgraphs or on one of their own induced subgraphs can be solved exactly in polynomial time. In [16], Warrier *et al.* demonstrate the feasibility of using such a partition on some (but not all) graphs, judging the performance of their branch-and-price approach to be fair but not excellent on those graphs under this partition scheme.

We have also demonstrated how the RMP of the branch-and-price method in [16] can be tightened and reduced in size by constructing clique constraints. Last, we have given a method for initializing the sets of known extreme points for use by RMP.

CHAPTER III

COMBINATORIAL BRANCH-AND-BOUND APPROACHES FOR THE MWIS PROBLEM

A. Introduction

Clique covers have been used to provide upper bounds in successful branch-and-bound solvers for both the MIS and MWIS problems. We describe herein three branch-and-bound algorithms based on clique covers. The first uses a cover-creation routine that is procedurally like those in [5] by Balas and Yu and [6] by Balas and Xue, but whose clique covers borrow structural features from those created by the algorithm in [27]. The second is a more direct improvement on Babel's algorithm that significantly reduces the run time required on integer-weighted graphs and generalizes to the case of real-valued weights. The third is a hybrid of the other two; it yields the best known results on some graphs.

The methods and conclusions of the three papers noted above will be important to our discussion, so we will explain their work in some detail. Section C summarizes these previous studies, and Section D contains our analysis of that strand of research, including some experimental work. Sections E and F describe our contributions that arose from that analysis, and Section G relates our subsequent computational results. Our conclusions from these new computations are given in Section H. First we will address some preliminary matters.

B. Preliminaries

For basic terminology and an introduction to graph theory, we refer the reader to [28]. All graphs in this chapter are finite and simple. We assume in this chapter that the vertex and edge sets of a graph named G are named V and E, respectively. The set of all neighbors in G of a vertex v is denoted $N_G(v)$, and its set of non-neighbors (namely $V \setminus N_G(v) \setminus \{v\}$) is denoted $\bar{N}_G(v)$; we will dispense with subscripts when the meaning is clear. We may associate with a graph G a weight function $w : V \to R$; for our purposes, the weights will be restricted to the positive reals. For every $U \subseteq V$, we denote by $w(U)$ the sum of the weights of its members, with $w(\varnothing) = 0$. The maximum cardinality of an independent set of G is denoted $\alpha(G)$. The maximum value of $w(S)$ among all independent sets S of G is denoted $\alpha_w(G)$.

A **clique** of graph G is a set of vertices $K \subseteq V$ such that all pairs of vertices in K are adjacent. Cliques of G are independent sets of $\bar{G}$, so the above problems are equivalent to the maximum clique problem and the maximum weight clique problem. The size of a maximum clique and the total weight of a maximum weight clique in G are denoted by $\omega(G)$ and $\omega_w(G)$. For an introduction to all these problems, we refer the reader to [2], which addresses them as clique problems.

A **clique cover** of graph G is a collection of cliques $K_1, \ldots, K_t$ such that $V = \bigcup_{i=1}^{t} K_i$. The least such t for G is denoted $\phi(G)$. The complementary notion — an analogous collection of independent sets — is called a **coloring**. The minimum number of sets in a coloring for G is denoted $\chi(G)$.

A **weighted clique cover** of G is a collection of cliques $J_1, \ldots, J_s$ together with real numbers $W_1, \ldots, W_s$ such that for every $v \in V$,

$$\sum_{i : v \in J_i} W_i \geq w(v).$$

We call the W_i the weights of their respective cliques (not to be confused with $w(J_i)$), and we call $\sum_{i=1}^{s} W_i$ the weight of the weighted clique cover. The least weight of a weighted clique cover of G is denoted $\phi_w(G)$. The complementary notion — an analogous collection of independent sets and weights — is called a **weighted coloring**, and the corresponding minimum weight for G is denoted $\chi_w(G)$.

Clique covers and weighted clique covers of G provide upper bounds on $\alpha(G)$ and $\alpha_w(G)$. The intersection of any independent set of G and any clique of G is empty or a singleton, thus in keeping with the above notation, these inequalities hold:

$$\alpha(G) \le \phi(G) \quad \le t$$
$$\alpha_w(G) \le \phi_w(G) \le \sum_{i=1}^{s} W_i.$$

Analogous inequalities hold for colorings and clique numbers

C. Previous Work

1. Balas and Yu

A graph G is **perfect** if, for each of its induced subgraphs $G[H]$, $\alpha(G[H]) = \phi(G[H])$. An equivalent condition is that $\omega(G[H]) = \chi(G[H])$ for each $G[H]$. Moreover, the complement of every perfect graph is itself perfect. Perfect graphs are of particular interest to us because polynomial-time algorithms exist to solve the MIS and MWIS problems on them.

A graph is **chordal** if all its cycles of length at least four have a chord. Chordal graphs (and their complements) are perfect. Every non-empty graph has a chordal induced subgraph since every graph on three vertices or fewer is chordal.

Based on an algorithm in [29], Balas and Yu developed in [5] a polynomial-time algorithm to find a vertex-maximal chordal induced subgraph of a given graph; the algorithm simultaneously finds a maximum clique of that subgraph. They also developed a polynomial-time algorithm to create an optimal coloring of a chordal graph. Analogous, complementary methods produce optimal clique covers and MISs of graphs whose complements are chordal.

Most importantly, they devised the following branching strategy that has been used in many subsequent research efforts:

1. For a graph G and an independent set $S \subseteq V$, find $U \subseteq V$ for which it is known that $\alpha(G[U]) \leq |S|$.
2. Order the vertices of $V \setminus U$ as $x_1, \ldots, x_k$.
3. For each i, let $V_i = \bar{N}(x_i) \setminus \{x_j : j < i\}$, and find a MIS S_i in $G[V_i]$.

Either S or one of $S_1 \cup \{x_1\}, \ldots, S_k \cup \{x_k\}$ is a MIS for G. The same method, of course, can be applied to the problem on each $G[V_i]$. Repeated application will yield MIS problems on induced subgraphs of the form $G[V \setminus (I \cup X)]$, where I is a set of forcibly included vertices (from the successive choices of x_i in Step 3 above, whereby I is an independent set) and where X is a set of forcibly excluded vertices (from the successive restrictions of the problem to $\bar{N}(x_i) \setminus \{x_j : j < i\}$ in Step 3). If an independent set S solves such a problem, then the independent set $I \cup S$ may solve the problem at the root node of the branch-and-bound tree. We will refer to this as the **Balas-Yu Branching Scheme**.

Balas and Yu put these pieces together as follows. First, find a vertex-maximal induced subgraph $G[T]$ such that $\bar{G}[T]$ is chordal, along with a MIS S of $G[T]$. Next, since $\phi(G[T]) = |S|$, optimally cover $G[T]$ with cliques $K_1, \ldots, K_{|S|}$. Clearly, each K_i contains one member of S. Then sequentially add as many vertices $v \in V \setminus T$ as possible (in an arbitrary sequence) to any of these cliques containing only vertices of $N(v)$, yielding cliques $\hat{K}_1, \ldots, \hat{K}_{|S|}$ (where $K_i \subseteq \hat{K}_i$ for each i). If we let $U = \bigcup_{i=1}^{|S|} \hat{K}_i$, then $\hat{K}_1, \ldots, \hat{K}_{|S|}$ is a clique cover of $G[U]$, and we see that $\alpha(G[U]) \leq \phi(G[U]) \leq |S|$ (since $S \subseteq U$, we actually have equality at each step). The sets S and U thus satisfy the requirements of Step 1 in the Balas-Yu Branching Scheme, so apply it. We will call this procedure for coloring and branching the **Chordal Method** of Balas and Yu.

Balas and Yu deemed the Chordal Method computationally expensive; it requires $O(|V| + |E|)$ time. They proposed a cheaper method to produce a set U and an independent set S to which their Branching Scheme would apply. Given an independent set S in G (in practice, the largest known), find disjoint cliques $K_1, \ldots, K_{|S|}$ and let $U = \bigcup_{i=1}^{|S|} K_i$. The method for finding the cliques is the same as applying the greedy clique-covering extension of the Chordal Method to a collection of $|S|$ empty sets. The cliques K_i cover $G[U]$, so $\alpha(G[U]) \leq \phi(G[U]) \leq |S|$, and the Balas-Yu Branching Scheme applies. We will call this procedure for creating a clique cover and branching the **Greedy Method** of Balas and Yu. This method also requires $O(|V|+|E|)$ time, but dispenses with finding a vertex-maximal chordal induced subgraph of $\bar{G}$, resulting in a smaller constant coefficient for $(|V| + |E|)$.

We call the following overall algorithm the **Balas-Yu Algorithm**. It corresponds to the variant called TC4 in [5]. The root node of the branch-and-bound tree corresponds to the MIS problem on G itself. Among all unsolved MIS problems on the induced subgraphs $G[V \setminus (I \cup X)]$, the algorithm chooses one with $|I|$ maximum. It then decides whether $|V \setminus X| \leq |S|$, in which case the problem is discarded since it cannot produce an independent set larger than S. Apart from that case, it applies the Chordal Method to $G[V \setminus (I \cup X)]$ if $|I| = |S|$ and applies the Greedy Method otherwise.

The computational results in [5] demonstrate that the Balas-Yu Algorithm handily outperforms its ablest predecessor, the algorithm in [4] by Bron and Kerbosch.

2. Balas and Xue

Balas and Xue [6] extended the Balas-Yu Algorithm to the weighted case. Chordal graphs remain useful in the weighted case, because, again, the complement of a chordal graph is perfect, and, for a perfect graph G, any $H \subseteq V$ satisfies $\alpha_w(G[H]) = \phi_w(G[H])$ for all $w : V \to R$. By extending the coloring algorithm in [5], Balas and Xue developed a polynomial-time algorithm to find an optimal weighted coloring of a chordal graph. Their simple generalization of the algorithm in [5] to find a vertex-maximal chordal induced subgraph finds a maximum weight clique of the subgraph while constructing the subgraph. They also generalized to the weighted case both the clique-cover extension procedure in [5] and the Balas-Yu Branching Scheme. The generalization of the branching scheme is straightforward: in Step 1, replace "$\alpha(G[U]) \leq |S|$" with "$\alpha_w(G[U]) \leq w(S)$", and in Step 3, replace "maximum" with "maximum weight." We will refer to this as the Balas-Yu Branching Scheme as well, since the generalization is so straightforward, and context will make clear which version (weighted or unweighted) we are referring to.

At each branch-and-bound node, Balas and Xue undertake a process that is mostly analogous to the Balas-Yu Algorithm. For a graph G with weight function w, find a vertex-maximal induced subgraph $G[T]$ such that $\bar{G}[T]$ is chordal, and simultaneously find a MWIS S of $G[T]$. Next, find a weighted clique cover of $G[T]$ consisting of cliques $K_1, \ldots, K_t$ and weights $W_1, \ldots, W_t$ such that the sum of all the weights is $\phi_w(G[T])$. Greedily add vertices of $V \setminus T$ to these cliques to create cliques $\hat{K}_1, \ldots, \hat{K}_t$ such that if $U = \bigcup_{i=1}^{t} \hat{K}_i$, then the sets $\hat{K}_1, \ldots, \hat{K}_t$ and the weights $W_1, \ldots, W_t$ are a weighted clique cover of $G[U]$. Since the weights are the same for both weighted clique covers, $\phi_w(G[U]) = \phi_w(G[T])$, whereby S is a MWIS for $G[U]$, so apply the Balas-Yu Branching Scheme. We will call this procedure for weighted coloring and branching the **Weighted Chordal Method** of Balas and Xue.

Like Balas and Yu in [5], Balas and Xue in [6] elected to apply this Weighted Chordal Method to only some nodes of the branch-and-bound tree (their account seems ambivalent, however; see pages 218–219 of their paper). Although they do not state explicitly the conditions for applying it, we assume that they are similar to those in the Balas-Yu Algorithm (probably they chose the obvious weighted analog). Also, they do not explain their method for branch-set construction for nodes at which they do not apply the Weighted Chordal Method. For this chapter, we assume that their procedure is similar to the one below. At each step, it creates a clique, assigns it a weight, and updates the amount of uncovered weight each vertex has. It also keeps track of vertices that should not be considered for membership in the cliques being generated by reason of their being fully covered already or being uncoverable with the remaining available weight.

1. Take a heaviest known independent set S of G.

2. Let $V_1 = V$, $w_1 = w$, and $i = 1$.

3. Find a maximal clique K_i of $G[V_i]$.

4. Let $W_i = \min\{w_i(v) : v \in K_i\}$.

5. Let $w_{i+1}(v) = w_i(v) - W_i$ for $v \in K_i$, and let $w_{i+1}(v) = w_i(v)$ otherwise.

6. Let $U_i = \{v \in V_i : w_{i+1}(v) = 0\}$, $Z_i = \left\{v \in V_i : w_{i+1}(v) > w(S) - \sum_{j=1}^{i} W_j\right\}$, and $V_{i+1} = V_i \setminus (U_i \cup Z_i)$.

7. If V_{i+1} is not empty, increment i and go to Step 3.

At the end of this procedure, $K_1, \ldots, K_i$ with weights $W_1, \ldots, W_i$ form a weighted clique cover of $G[U_1 \cup \cdots \cup U_i]$. The Balas-Yu Branching Scheme then applies. We will call this procedure for weighted coloring and branching the **Weighted Greedy Method**.

While the Weighted Greedy Method is not explicitly specified in [6], it is a straightforward extension of the Greedy Method of Balas and Yu. For G and a constant w, it would produce a weighted clique cover of G equivalent to the (unweighted) clique cover produced by the Greedy Method when applied to G with no weight function, assuming the cliques in Step 3 of the Weighted Greedy Method are found in the same way as those found in the Greedy Method. Note that the use of Z_i in Step 6 would be superfluous for a constant weight function w. We have included it in the Weighted Greedy Method since, for a non-constant w, it provides slightly better covers at little additional cost, and a similar technique is employed in one of our own methods (see Section 1).

When combined with a problem-selection and fathoming process analogous to that in [5], these weighted clique cover methods yield the **Balas-Xue Algorithm**. The computational results in [6] demonstrate that it outperforms its ablest predecessor, a weighted version of the algorithm in [9].

3. Babel

Babel [27] introduced a new method for generating weighted clique covers for graphs whose vertex weights are positive integers. Because of the restriction on weights, this method can apply an equivalent definition of weighted clique cover: a weighted clique cover of weight t for the graph G with integer-valued weight function w is a collection of cliques $K_1, \ldots, K_t$ such that for every $v \in V$,

$$\left|\left\{K_j : v \in K_j\right\}\right| \geq w(v).$$

Note that W instances of the same clique according to this definition would correspond to a clique with weight W in our previous definition of weighted clique cover.

Given a collection $\{K_1, \ldots, K_t\}$ of cliques in G, the **generalized clique degree** (GKD) of a vertex $v \in V$ is

$$\left|\bigcup_{u \in \bar{N}(v)} \left\{K_j : u \in K_j\right\}\right|.$$

If those cliques contain only vertices of some $U \subseteq V$ and are a weighted clique cover for $G[U]$, then a vertex $v \in V \setminus U$ with a GKD greater than $t - w(v)$ cannot be added to enough cliques in the collection for $G[U \cup \{v\}]$ to be covered. GKD of vertices in $V \setminus U$ thus indicates the least number of cliques that must be added to the collection for it to become a weighted clique cover of G.

GKD is a weighted (and complementary) analog to the quantity called **saturation degree** in [30]. The method in [27] for generating weighted clique covers, which we will refer to as the **Babel Method**, is likewise an analog to the coloring method in [30].

At every step, the Babel Method maintains a list of cliques. Each step involves selecting a vertex and adding it to some cliques in the list. At the end of each step, the cliques are a weighted clique cover of the subgraph induced by the vertices selected thus far.

Begin each step by choosing a vertex v having maximum GKD among all uncovered vertices, breaking ties by choosing heavier vertices. Assign to W the weight $w(v)$; this represents the uncovered weight of v. Inspect the cliques in the list in order. If all members of the clique are neighbors of v, then add v to K and decrement W. If $W = 0$, end the current step. If $W > 0$, inspect the next clique in the list. If $W > 0$ after all cliques have been inspected, then add W instances of the clique $\{v\}$ to the end of the list and end the current step. Note that since vertices are selected by GKD, the members of some maximal independent set S are the first vertices covered.

Babel also noted, however, that some vertices can be eliminated from consideration after covering the graph. For each $v \in V$, let $\mathcal{K}(v)$ be the collection of cliques in the weighted clique cover that contain v or any member of $\bar{N}(v)$. $\mathcal{K}(v)$ is a weighted clique cover for $G[\{v\} \cup \bar{N}(v)]$, so if $|\mathcal{K}(v)| \leq w(S)$, then $\alpha_w(G[\{v\} \cup \bar{N}(v)]) \leq w(S)$, and v belongs to no independent set of G heavier than S. If we let $D = \{v \in V : |mathcalK(v)| \leq w(S)\}$, then $\alpha_w(G - D) > w(S)$ if and only if $\alpha_w(G) > w(S)$. Thus, instead of branching based on our weighted clique cover, we can discard it, retain S, and begin again by covering $G - D$. Babel reports in [27] that such elimination and re-covering is often advantageous, sometimes eliminating the need for branching altogether.

When needed, branching is performed as follows. After completing the Babel Method, index the cliques in the weighted clique cover as $K_1, \ldots, K_t$ according to their order in the list. For each $v \in V$, define $r(v)$ as the greatest $j \in \{1, \ldots, t\}$ such that $v \in K_j$. Order the vertices in V as $v_1, \ldots, v_n$ so that r is non-increasing on the sequence. Let s be greatest in $\{1, \ldots, n\}$ such that $r(v_s) > w(S)$. Note, then, that $\{v_{s+1}, \ldots, v_n\}$ is a valid choice for U in the Balas-Yu Branching Scheme. If we maintain the order $v_1, \ldots, v_s$ for the vertices to be branched on, then branching yields vertex sets $V_1, \ldots, V_s$. Then for every $i \in \{1, \ldots, s\}$, $r(v_i) \geq r(v)$ for all $v \in V_i$.

Since no $v \in V_i$ belongs to any clique in the weighted clique cover to which v_i also belongs, we conclude that the weighted clique cover induces a weighted clique cover of weight $r(v_i) - w(v_i)$ on $G[V_i]$ and one of weight $r(v_i)$ on $G[\{v_i\} \cup V_i]$. We retain $r(v_i)$ since it provides these upper bounds on $\alpha_w(G[V_i])$ and $\alpha_w(G[\{v_i\} \cup V_i])$. If we then find an independent set of G with weight at least $r(v_i)$ before solving the sub-problem on $G[V_i]$, we can discard the sub-problem. We call this combination of the Babel Method with the Balas-Yu Branching Scheme the **Babel Algorithm**.

Babel reported in [27] significant improvements over the performance of the Balas-Xue Algorithm in most cases, especially on low-density graphs.

D. Our Analysis and Discoveries

We first note that the Balas-Yu Algorithm seldom applies the Chordal Method. At the root node of the branch-and-bound tree both the best known independent set S and the set of forcibly included vertices I are empty. They satisfy the condition $|I| = |S|$ under which the Chordal Method is applied (see the end of Section 1). The Chordal Method is otherwise used only at a node rather deep in the branch-and-bound tree (assuming that a reasonably good clique was found at the root node) when $|I| = |S| > 0$ (whereby I is also a best known independent set) and yet $V \setminus (I \cup X)$ is nonempty (whereby any of its vertices could be added to I to form a new best known independent set). So every time the Chordal Method is used, the best known independent set is guaranteed to be replaced. Yet in those very situations, an improving independent set could easily be found without using the Chordal Method.

Now, if, at the root node, the Chordal Method found a MIS S of G but could not cover G with $|S|$ cliques, causing the Balas-Yu Algorithm to branch, then the

Chordal Method would never again be used. The same would be true if we were to supplement the Balas-Yu Algorithm by first using a heuristic method to find some large independent set and happened to find a MIS. So the power behind the Balas-Yu Algorithm seems to be its Greedy Method and the Balas-Yu Branching Scheme; the Chordal Method appears to serve primarily as a heuristic. If our assumptions in Section 2 about the Balas-Xue Algorithm are correct, then its Weighted Chordal Method would also seem to serve primarily as a heuristic. This is suggested by some of the results presented in [5] (see their Table 5.1), but it is clearly demonstrated by an experiment of ours that we now describe.

We implemented the Balas-Xue Algorithm (under the assumptions mentioned in Section 2, whereby, for constant weight functions, our implementation reduces to an implementation of the Balas-Yu Algorithm) and two variants: one that uses the Weighted Chordal Method at every node of the branch-and-bound tree and another that likewise uses the Weighted Greedy Method. We call these adaptations the **Chordal Variant** and the **Greedy Variant**. We started the Greedy Variant by applying a modest greedy heuristic; see Section 1 for details. For constant w, all three implementations reduce to variants of the Balas-Yu Algorithm. We ran these three algorithms (six, counting the unweighted versions) to solve the MIS and MWIS problems on several graph instances.

Table II shows results obtained by solving the MIS problem on some graphs from the Second DIMACS Implementation Challenge [25]; the times are in seconds, and an asterisk (*) denotes instances in which computer memory was exhausted. Table III shows results obtained by solving the MWIS problem on some uniform random graphs with uniformly distributed vertex weights; see Section G for details about these graphs. Each line of Table III gives the average times and numbers of branch-and-bound nodes across a sample of 10 graphs with the same edge probability p. The range of edges in the graphs is given in the column under $|E|$.

Table II. The Balas-Yu algorithm and its variants

				Balas-Yu		Greedy		Chordal	
Graph	$\|V\|$	$\|E\|$	$\alpha(G)$	Time	Nodes	Time	Nodes	Time	Nodes
MANN_a9	45	72	16	0.1	477	0.1	507	15.5	62,204
brock200_1	200	5,066	21	*	>700,000	705.6	645,467	*	>500,000
brock200_2	200	4,024	12	5.1	4,179	4.6	3,915	50.4	65,573
brock200_3	200	7,852	15	35.6	33,014	37.6	35,565	*	>300,000
brock200_4	200	6,811	17	60.2	51,178	78.0	72,894	*	>400,000
c-fat200-1	200	18,366	12	<0.1	13	<0.1	5	<0.1	12
c-fat200-2	200	16,575	24	<0.1	2	<0.1	1	<0.1	2
c-fat200-5	200	8,473	58	0.1	30	0.1	28	0.2	30
c-fat500-1	500	120,291	14	0.1	11	<0.1	1	0.1	11
c-fat500-2	500	115,611	26	0.2	11	0.1	1	0.1	11
c-fat500-5	500	101,559	64	0.1	3	0.1	1	0.1	3
c-fat500-10	500	78,123	126	0.3	3	0.1	1	0.3	3
hamming8-2	256	1,024	128	28.1	8,691	29.2	8,751	*	>400,000
hamming8-4	256	11,776	16	3.1	1,744	2.5	1,505	*	>300,000
johnson8-2-4	28	168	4	<0.1	30	<0.1	22	<0.1	76
johnson8-4-4	70	560	14	<0.1	37	<0.1	26	4.0	10,458
johnson16-2-4	120	1,680	8	74.8	218,683	72.0	218,387	*	>600,000
keller4	171	5,100	11	7.4	9,516	10.8	13,742	*	>400,000
p_hat300-1	300	33,917	8	3.1	1,716	3.1	1,719	11.7	12,807
p_hat300-2	300	22,922	25	310.5	135,703	345.9	149,974	*	>250,000

Table III. The Balas-Xue algorithm and its variants

			Balas-Xue		Greedy		Chordal	
$\|V\|$	p	$\|E\|$	Time	Nodes	Time	Nodes	Time	Nodes
100	0.5	2,428–2,502	0.2	305.6	0.2	293.8	1.1	2,554.1
100	0.4	1,945–2,039	0.4	644.8	0.4	547.9	4.6	10,066.3
100	0.3	1,445–1,524	1.4	2,303.5	1.4	2,229.0	27.2	54,415.1
100	0.2	953–1,049	5.9	9,410.4	6.2	9,582.9	203.1	335,283.7
100	0.1	455–518	33.0	50,314.3	32.8	47,946.8	*	>800,000
200	0.5	9,851–10,052	6.3	4,130.0	6.2	4,022.7	51.4	67,195.2
200	0.4	7,839–8,095	29.0	19,101.0	29.6	19,542.1	*	>350,000
200	0.3	5,887–6,055	183.8	117,786.7	193.6	124,208.0	*	>400,000
200	0.2	3,939–4,057	*	>500,000	*	>500,000	*	>400,000

We take a brief aside here to note that all three key papers report results from too few random graphs. Balas and Yu [5] generated only one random graph for each set of graph-generation parameters. Balas and Xue [6] generated only two graphs per set, and Babel [27] used those same graph instances. When we applied our implementations of their algorithms to ten graphs per set of graph-generation parameters, we were unable to replicate the average branch-and-bound tree sizes that they reported (run times, of course, depend on machine speed). Our basis for comparison, then, will be our own implementations of the algorithms applied to the random graphs that we ourselves generated. Our results differ somewhat from theirs, accordingly.

Balas and Yu implemented something similar to the Greedy Variant (again, some of the details are not specified in [5]) and found it to be as good as or better than the Balas-Yu Algorithm for graphs with relatively small MISs but considerably worse for graphs with larger MISs. Our implementation of the Greedy Variant outperformed our implementation of the Balas-Yu Algorithm slightly in most cases that pose any challenge. The initial heuristic we employed probably accounts for the similarity in their performances. Balas and Yu also implemented the Chordal Variant and noted its long run times (as stated earlier, the reason they developed the Greedy Method). However, we note that, beyond requiring long run times, our implementation of the Chordal Variant produces a much larger branch-and-bound tree than either our implementation of the Balas-Yu Algorithm or the Greedy Variant. Similar results hold for the weighted case, as seen in Table III.

There are two reasons for this. The first has to do with the independent sets that the Chordal Method finds in nodes below the root of the branch-and-bound tree and their effect on the clique covers generated at those nodes. Consider the MIS problem on G and a sub-problem with inclusion and exclusion sets I and X. Suppose the largest known independent set of G is S, and $|I| < |S|$. Upon finding a vertex-maximal $T \subseteq V \setminus (I \cup X)$ such that $\bar{G}[T]$ is chordal and finding a MIS S_T of $G[T]$, we are not guaranteed that $|I \cup S_T| \geq |S|$, even if S_T is also a MIS of $G[V \setminus (I \cup X)]$. In that case, we would produce a clique cover having only $|S_T|$

cliques, even though a clique cover with $|S|-|I|$ cliques would be legitimate to use for branching. We could, therefore, add any $|S|-|I \cup S_T|$ cliques of $G[V \setminus (I \cup X)]$ to our clique cover before branching, reducing the number of children produced by the current sub-problem and the size of the whole branch-and-bound tree. An analogous savings can be achieved for the weighted case and the Weighted Chordal Method.

So, we added this feature to both the Balas-Yu and Balas-Xue Algorithms, with extra cliques being chosen greedily. The results of this **Modified Chordal Variant** are given in Tables IV and V. The times and tree sizes are considerably improved on the challenging problems, but they still do not approach those for the Greedy Variant, save on p_hat300-2.

Table IV. The Modified Chordal Variant of the Balas-Yu algorithm

Graph	$\|V\|$	$\|E\|$	$\alpha(G)$	Time	Nodes
MANN_a9	45	72	16	5.1	20,393
brock200_1	200	5,066	21	*	>400,000
brock200_2	200	10,024	12	13.9	24,622
brock200_3	200	7,852	15	90.8	131.625
brock200_4	200	6,811	17	182.5	206,220
c-fat200-1	200	18,366	12	<0.1	12
c-fat200-2	200	16,575	24	<0.1	2
c-fat200-5	200	8,473	58	0.2	30
c-fat500-1	500	120,291	14	0.1	11
c-fat500-2	500	115,611	26	0.1	11
c-fat500-5	500	101,559	64	0.1	3
c-fat500-10	500	78,123	126	0.3	3
hamming8-2	256	1,024	128	2.4	51
hamming8-4	256	11,776	16	58.0	36,021
johnson8-2-4	28	168	4	<0.1	76
johnson8-4-4	70	560	14	0.2	350
johnson16-2-4	120	1,680	8	*	>600,000
keller4	171	5,100	11	33.0	50,471
p_hat300-1	300	33,917	8	4.4	6,225
p_hat300-2	300	22,922	25	153.9	67,045

This under-performance indicates to us a second reason for the inferiority of the original Chordal Variant: there is a fundamental difference in the quality of weighted clique covers (and, thus, branch sets) produced by the Chordal and

Table V. The Modified Chordal Variant of the Balas-Xue algorithm

$\lvert V\rvert$	p	$\lvert E\rvert$	Time	Nodes
100	0.5	2,428–2,502	0.3	644.4
100	0.4	1,945–2,039	0.8	1,511.5
100	0.3	1,445–1,524	2.3	3,062.6
100	0.2	953–1,049	11.6	11,186.4
100	0.1	455–518	43.7	21,616.5
200	0.5	9,851–10,052	6.8	8,525.3
200	0.4	7,839–8,095	34.1	35,971.5
200	0.3	5,887–6,055	235.6	171,087.6
200	0.2	3,939–4,057	*	>400,000

Greedy Methods. Having now given these methods the same upper bound to use in constructing a weighted clique cover, the two methods still fare differently.

The Babel Algorithm especially indicates the potential for varying quality in weighted clique covers. Its clique covers are generally better than those of any of the above variants of the Balas-Yu and Balas-Xue Algorithms, resulting in considerably smaller trees for many problem instances; Section G gives some examples. The Babel Method is not without its weaknesses, however.

One weakness of the Babel Method is the computational expense of producing its weighted clique covers. A rough analysis of the Babel Method indicates that its run time is $O(W\,|V|^2)$, where W is the maximum weight in the graph. Recall that the run time of the Greedy Method is $O(|V| + |E|)$, which is itself $O(|V|^2)$. So a large value of W could (depending on the distribution of weights) offset any difference between the constant coefficients of these big-O measures. In practice, we find that the Babel Method almost always takes longer than the Greedy Method, even for W as low as 2.

This weakness is compounded once a MWIS of G has been found but not proved maximum. The upper bounds computed for all newly-created sub-problems will not be used to fathom them. Therefore, covering the vertices that eventually fall in $V \setminus U$ serves no purpose but exacts considerable cost.

Another weakness of the Babel Method is exposed when we do indeed consider large values of W: namely, its lack of scalability with respect to vertex weights. If the Babel Method constructs q cliques to create a weighted clique cover for G with weight function w, then it constructs $p \times q$ cliques when the weight function is changed to $p \times w$ for some integer p. Since the cost of the Babel Method dominates the overall computational cost of the Babel Algorithm, increasing the weight function by a factor of p results in a nearly p-fold increase in the run time of the Babel Algorithm. Changing the weight function this way would have no effect on the Balas-Xue algorithm (assuming no increase in computer storage is required to represent the new vertex weights). In general, the Babel Algorithm fares better against the Balas-Xue Algorithm on graphs with small vertex weights than on graphs with large vertex weights.

Another weakness of the Babel Method is its inability to accommodate real-valued weight functions. Obviously, it can accommodate rational vertex weights by changing them to integer weights, but the multiplication required for this change incurs the aforementioned cost for large vertex weights.

In the next section, we will demonstrate that the weaknesses of these algorithms can be remedied. We will modify the Greedy Method to more nearly replicate the weighted clique cover successes of the Babel Method while maintaining the low cost of the Greedy Method. With a superior data representation, we will overcome the chief weaknesses of the Babel Method without sacrificing the high quality of its weighted clique covers.

E. New Clique Cover Methods

1. Method A

The Babel Method spends its first steps adding the vertices of a maximal independent set to its weighted clique cover. We found that the Weighted Greedy Method is greatly improved by forcing a heavy independent set to be included in the subgraph $G[U]$ for which the Weighted Greedy Method finds a weighted clique cover. We call this approach our **Method A**. Step 3 forces $S \subseteq U_1 \cup \cdots \cup U_i$ by the end of the procedure.

1. Take a heaviest known independent set S of G.
2. Let $V_1 = V$, $K_1 = \varnothing$, $w_1 = w$, and $i = 1$.
3. Find a maximal clique $\hat{K}_i$ of $G[V_i]$. If $S \cap K_i$ is empty and $S \cap V_i$ is not empty, ensure that a vertex of S is in $\hat{K}_i$. Otherwise, ensure that $K_i \subseteq \hat{K}_i$.
4. Let $W_i = \min\{w_i(v) : v \in \hat{K}_i\}$.
5. Let $w_{i+1}(v) = w_i(v) - W_i$ for $v \in \hat{K}_i$, and let $w_{i+1}(v) = w_i(v)$ otherwise.
6. Let $U_i = \{v \in V_i : w_{i+1}(v) = 0\}$, let $Z_i = \left\{v \in V_i : w_{i+1}(v) > w(S) - \sum_{j=1}^{i} W_j\right\}$, and let $V_{i+1} = V_i \setminus (U_i \cup Z_i)$.
7. If V_{i+1} is not empty, then let $K_{i+1} = \hat{K}_i \cap V_{i+1}$, increment i and go to Step 3.

At the end of this procedure, $K_1, \ldots, K_i$ with weights $W_1, \ldots, W_i$ form a weighted clique cover of $G[U_1 \cup \cdots \cup U_i]$.

Of course, for constant w, Method A reduces to a method for generating clique covers. Its overhead can thus be reduced substantially. Steps 4 and 5 are unnecessary. Every U_i is simply $\hat{K}_i$. Computing Z_i is unnecessary. Every K_i is empty, simplifying the logic of Step 3.

At a branch-and-bound node with forcibly included vertices I and forcibly excluded vertices X, the heaviest known independent set S in Step 1 is an independent set of $G[V \setminus (I \cup X)]$. But $w(S \cup I)$ might fall short of the weight of some heaviest known independent set S_G in G. In that case, we can replace $w(S)$ in Step 6 with $w(S_G) - w(I)$. It is only in this case that we can have $S \cap V_i = \varnothing$ in Step 3 and thus have cliques $\hat{K}_i$ containing no vertex of S.

The upper bound computed for the child sub-problem corresponding to $x_i \in V \setminus (I \cup X \cup U)$ is simply the lesser of the upper bound for the parent sub-problem and

$$w(S) + w\left(\left\{x_j : j \geq i\right\}\right),$$

though this could easily be made better by accounting for such x_j that are neighbors of x_i.

2. Method B

We amended the Babel Method to produce a clique-covering method that more efficiently accommodates non-constant weight functions and also accommodates arbitrary real-valued weight functions. We call it **Method B**. For graphs with integer vertex weights, it constructs a weighted clique cover that is equivalent to that produced by the Babel Method.

Method B uses the original definition of weighted clique cover that we gave in Section B. As a result, we provide this equivalent definition of GKD. Given a collection $\{K_1, \ldots, K_t\}$ of cliques in G and a set of real weights $\{W_1, \ldots, W_t\}$, the GKD of a vertex $v \in V$ is the sum of weights corresponding to cliques containing non-neighbors of v. Specifically, if we define

$$Q(v) = \{i \in \{1, \ldots, t\} : K_i \cap \bar{N}(v) \neq \emptyset\},$$

the GKD of v is then $\sum_{i \in Q(v)} W_i$.

At every step, Method B maintains a list of cliques and a corresponding list of weights. Each step involves selecting a vertex, adding it to some cliques in our list, and adjusting their weights, if necessary. At the end of each step, the cliques and weights are a weighted clique cover of the subgraph induced by the vertices selected thus far.

Begin each step by choosing a vertex v having maximum GKD among all uncovered vertices, breaking ties by choosing heavier vertices. Assign to W_v the weight $w(v)$. Inspect the cliques in the list in order. If all members of a clique K are neighbors of v, then compare W_v to the weight W of K. If $W < W_v$, add v to K, reduce W_v by W, and inspect the next clique in the list. If $W = W_v$, add v to K and end the current step. If $W > W_v$, insert a second instance $\hat{K}$ of K into the list after K, add v to K, assign W_v as the weight of K, assign $W - W_v$ as the weight of $\hat{K}$, and end the current step. If $W_v > 0$ after all cliques have been inspected, add the clique $\{v\}$ to the end of the list, give it weight W_v, and end the current step.

Let $K_1, \ldots, K_t$ be our final list of cliques, and let $W_1, \ldots, W_t$ be their weights. Define $s(v)$ as the greatest $j \in \{1, \ldots, t\}$ such that $v \in S_j$. Defining $r'(v)$ as

$$\sum_{i=1}^{s(v)} W_i,$$

we note that replacing $r(v)$ with $r'(v)$ in the upper bound computations of the Babel Method is valid.

F. Implementation Issues

1. The Branch-and-Bound Algorithms

Methods A and B are used in two branch-and-bound algorithms that we will call **Algorithms A and B**, respectively. In Section G, we compare these algorithms with their respective ancestors, the Balas-Xue Algorithm and the Babel Algorithm (but see the note on Algorithm A* in Section G). We also created a hybrid algorithm that we will call **Algorithm AB**. It employs Method B in the first three generations of the branch-and-bound tree (*i.e.*, for sub-problems on $G[V \setminus (I \cup X)]$ with $|I| \leq 2$) and Method A elsewhere. We will demonstrate in Section G that it often outperforms both Algorithms A and B. Since it is the best representative of our new work, we will compare it against the best general algorithm known for the MWIS problem, the algorithm of Östergård in [11]. The rest of this section gives further details on the implementation of the three algorithms.

When we choose sub-problems in any of our three algorithms, we select a sub-problem for which the known upper bound on $\alpha_w(G)$ is greatest. At the root node of the branch-and-bound tree, we assign $w(V)$ as the upper bound for the problem on G.

The ordering of the vertices in $V \setminus (I \cup X \cup U)$ before branching is already prescribed for Method B. The choice of order is important for Method A, but we did not investigate new methods. Based on the combined experiences of other researchers [9, 12, 15], we ordered those vertices in increasing order of the total weight of their neighbors.

Babel's technique of elimination and re-covering noted in Section 3 is used in Algorithm B and at the root node of Algorithm AB. We also developed a simplified version of this technique that does not require that the whole graph be covered. We use it in Algorithm AB when Method A is applied, though we do not use it in Algorithm A to allow a more direct comparison to the Balas-Xue Algorithm. Consider a graph G with weight function w and heaviest known independent set S. Suppose we know $U \subseteq V$ such that $\alpha_w(G[U]) \leq w(S)$, and we apply the Balas-Yu Branching Scheme to $V \setminus U$, resulting in the sets $I_1, \ldots, I_k$ and $X_1, \ldots, X_k$ of included and excluded vertices, respectively. If any vertex belongs to $\bigcap_{i=1}^{k} X_i$, then it belongs to no independent set of G heavier than S. Instead of continuing with this branching, we could replace the graph G with $\hat{G} = G - \bigcap_{i=1}^{k} X_i$. It can be shown that the Babel Method always produces an empty $\bigcap_{i=1}^{k} X_i$ because of its technique of elimination and re-covering. It therefore cannot use this technique as an enhancement. The same is true of Method B.

2. Heuristics

Our heuristic to find heavy independent sets is a best-in greedy heuristic. The heuristic uses the function

$$H_G(v) = \frac{w(v)}{1 + w(N_G(v))}$$

(defined even when $w(N_G(v)) = 0$) to assign a value to each vertex in G and selects a vertex $\hat{v}$ that maximizes H. It then removes all neighbors of $\hat{v}$ from consideration, computes $H_{G[\bar{N}(\hat{v})]}(v)$ for all $v \in \bar{N}(\hat{v})$, and selects another best vertex. It continues to pick vertices and compute a new H at every iteration until the selected vertices form a maximal independent set of G. Note that if w is constant, then $H_G(v)$ is simply $1/d(v)$, and selecting a vertex that maximizes H is the same as choosing a vertex of least degree. We use a complementary procedure to find heavy (and large) cliques (see Sections D and 1). This heuristic is applied to G to begin our Greedy Variant, as described in Section D. It is also applied in

Algorithm A to $G[V \setminus (I \cup X)]$ at each branch-and-bound node to provide the independent set S needed by Method A (see Section 1).

In Algorithms A and AB, before we begin to process the branch-and-bound tree, we apply this heuristic to several induced subgraphs of G. Letting k be the lesser of $|V|$ and $10 + \left\lfloor \frac{|V|}{50} \right\rfloor$, we choose k vertices $\{v_1, \ldots, v_k\}$ having the highest values of H_G. We then apply the heuristic to each $G[\{v_i\} \cup \bar{N}(v_i)]$ to initialize our heaviest known independent set. The choice of k is arbitrary, of course, but we find that this choice is usually much better than $k = 1$ and that larger values of k are rarely useful for such a simple initial heuristic. More sophisticated heuristics would certainly be valuable when solving difficult problems. We do not augment Algorithm AB with this initial heuristic to allow a more direct comparison with the Babel Algorithm.

G. Computational Results

All our algorithms were implemented in C and run on a computer with dual 2.2-GHz Athlon processors and 2 GB memory (the code was not specialized for multiple processors).

To generate a random graph like those that we have used, first specify the vertex set and a probability p. For each possible edge in the graph, generate a random deviate r in $\mathrm{U}[0, 1]$ and include the edge in the graph if $r < p$. Then for each vertex, generate a random deviate in $\mathrm{DiscreteU}[1, 10]$ to be the weight of the vertex.

Instead of comparing Algorithm A to the Balas-Yu and Balas-Xue Algorithms themselves, we will compare it to improved versions of those algorithms. We demonstrated already that the Greedy Variant usually surpasses the Balas-Yu and Balas-Xue Algorithms in performance (see Section D). The Greedy Variant employs the Greedy Method within the branch-and-bound framework of the Balas-Yu and Balas-Xue Algorithms. If, instead, we use the Greedy Method along with our initial heuristic, upper bound technique, and problem-selection strategy, then we should obtain an algorithm that behaves, for the most part, like the Balas-Yu and Balas-Xue Algorithms (which use the Greedy Method at most nodes), but that enjoys the benefits of these low-cost improvements. We call the result **Algorithm A*** (whether applying it to weighted or unweighted graphs). It provides a good point of comparison with Algorithm A since the only difference between these two is in their branch-set construction methods.

Algorithm A* is almost always better than the Balas-Xue Algorithm and the Greedy Variant. The only case we found for which it performs substantially worse than the Balas-Xue Algorithm is p_hat300-2.

The computational results for Algorithms A and A* on the complements of some DIMACS graphs and some random graphs are given in Tables VI and VII, respectively. In Table VII, the times and numbers of nodes in each row are again averages across a sample of ten graphs, and the numbers under $|E|$ are again ranges. The random graphs of Table VII are the same as those of Tables III and V in Section D.

Table VI demonstrates that Algorithm A* outperforms Algorithm A only on johnson16-2-4 among the listed DIMACS graphs. In Table VII, we see that Algorithm A outperforms Algorithm A* on average for all classes of random graphs listed. In fact, Algorithm A outperformed Algorithm A* on every random graph instance except for two instances with $p = 0.4$.

We compared Algorithm B to the Babel Algorithm on the same random graphs as above and give the results in Table VIII. (Note that their performance

Table VI. Algorithms A and A* on DIMACS graphs

				A		A*	
Graph	$\|V\|$	$\|E\|$	$\alpha(G)$	Time	Nodes	Time	Nodes
MANN_a9	45	72	16	<0.1	119	0.1	507
brock200_1	200	5,066	21	454.0	409,818	*	>600,000
brock200_2	200	4,024	12	3.8	3,094	4.7	3,938
brock200_3	200	7,852	15	23.3	21,753	37.8	35,565
brock200_4	200	6,811	17	50.2	41,518	55.7	48,063
c-fat200-1	200	18,366	12	<0.1	5	<0.1	5
c-fat200-2	200	16,575	24	<0.1	1	<0.1	1
c-fat200-5	200	8,473	88	0.1	28	0.1	28
c-fat500-1	500	120,291	14	0.1	1	0.1	1
c-fat500-2	500	115,611	26	0.1	1	0.1	1
c-fat500-5	500	101,559	64	0.2	1	0.2	1
c-fat500-10	500	78,123	126	0.2	1	0.2	1
hamming8-2	256	1,024	128	<0.1	1	29.0	8,759
hamming8-4	256	11,776	16	1.4	771	2.6	1,505
johnson8-2-4	28	168	4	<0.1	22	<0.1	22
johnson8-4-4	70	560	14	<0.1	1	<0.1	26
johnson16-2-4	120	1,680	8	84.5	249,278	71.9	218,387
keller4	171	5,100	11	6.1	7,657	10.8	13,742
p_hat300-1	300	33,917	8	2.8	1,522	3.0	1,665
p_hat300-2	300	22,922	25	57.3	21,871	344.6	149,761

is almost identical for unweighted graphs, so their results on the DIMACS graphs are reported further below in Table X, where we compare them to the results of Algorithm AB.)

Table VIII demonstrates that Algorithm B outperforms the Babel Algorithm for all classes of random graphs listed. We also note that Algorithm A likewise outperforms the Babel Algorithm. Algorithm B produces the same number of branch-and-bound nodes for each problem as the Babel Algorithm, as it was

Table VII. Algorithms A and A* on weighted random graphs

			A		A*	
$\|V\|$	p	$\|E\|$	Time	Nodes	Time	Nodes
100	0.5	2,428–2,540	0.2	226.2	0.2	260.6
100	0.4	1,928–2,053	0.3	430.0	0.3	521.1
100	0.3	1,445–1,524	0.9	1,290.3	1.3	2,091.8
100	0.2	937–1,049	3.1	4,460.9	5.6	8,789.8
100	0.1	455–518	6.2	2,294.4	32.2	47,147.4
200	0.5	9,851–10,052	5.2	3,254.5	5.7	3,695.5
200	0.4	7,839–8,095	23.0	14,459.8	27.0	17,692.4
200	0.3	5,887–6,055	128.0	76,689.9	180.2	115,288.5
200	0.2	3,939–4,057	1,099.2	610,191.3	1,532.6	798,435.5

Table VIII. Algorithm B and the Babel algorithm on weighted random graphs

$\lvert V\rvert$	p	$\lvert E\rvert$	B Time	Babel Time	Nodes
100	0.5	2,428–2,540	0.1	0.2	74.6
100	0.4	1,928–2,053	0.2	0.4	131.3
100	0.3	1,445–1,524	0.4	1.3	286.0
100	0.2	937–1,049	1.1	4.2	627.8
100	0.1	455–518	1.8	8.5	692.5
200	0.5	9,851–10,052	3.0	7.7	1,021.2
200	0.4	7,839–8,095	10.3	30.4	2,795.1
200	0.3	5,887–6,055	48.1	163.3	10,351.6
200	0.2	3,939–4,057	451.3	1,705.6	72,300.1

designed to do, but takes approximately one-half to one-quarter the time that the Babel Algorithm does. This reduction factor depends on the distribution of vertex weights. A great variety of investigations are possible into the relationship between weight distribution and the difference in performance between these two algorithm, so we cannot be expansive here. Table IX, however, gives a taste of such an investigation. It shows the performance of each algorithm on graphs with vertex weights drawn from the discrete uniform distributions on the intervals listed. We again solved ten problems per set of parameters.

Table IX. The effect of weight distribution on Algorithm B and the Babel algorithm

$\lvert V\rvert$	p	$\lvert E\rvert$	Interval	B Time	Babel Time	Nodes
100	0.1	455–518	$[1, 10]$	1.8	8.5	692.5
100	0.1	468–524	$[1, 20]$	1.5	12.2	397.1
100	0.1	463–540	$[11, 20]$	9.9	95.1	3,791.9

Having demonstrated that Algorithms A and B are competitive with their ancestors, we now wish to consider their usefulness with respect to all existing algorithms. To that end, we will use Algorithm AB, the hybrid described in Section 1. Deep in the branch-and-bound tree for Algorithm AB, both Methods A and B can usually cover the induced subgraph $G[V \setminus (I \cup X)]$ well enough to avoid branching. By using the faster Method A below the root node, Algorithm AB saves time over Algorithm B. By using the usually higher-quality covers of Method B at the root node, it produces a smaller branch-and-bound tree than Algorithm A. Balancing these trade-offs between speed and cover quality in Algorithm AB results in a faster algorithm overall (but note that, for some graphs, even its branch-and-bound tree is smaller than those of either Algorithms A and B). Tables X and XI shows how Algorithm AB performs on the graphs we have encountered so far.

Table X. Algorithms B and AB on DIMACS graphs

				Babel/B		AB	
Graph	$\lvert V\rvert$	$\lvert E\rvert$	$\alpha(G)$	Time	Nodes	Time	Nodes
MANN_a9	45	72	16	<0.1	184	0.1	507
brock200_1	200	5,066	21	359.1	201,707	480.4	345,422
brock200_2	200	4,024	12	3.3	2,689	3.0	1,731
brock200_3	200	7,852	15	19.6	16,049	26.5	18,563
brock200_4	200	6,811	17	40.6	29,150	40.5	24,165
c-fat200-1	200	18,366	12	<0.1	1	<0.1	1
c-fat200-2	200	16,575	24	<0.1	1	<0.1	1
c-fat200-5	200	8,473	88	0.1	1	0.1	1
c-fat500-1	500	120,291	14	0.1	1	0.2	1
c-fat500-2	500	115,611	26	0.3	1	0.4	1
c-fat500-5	500	101,559	64	1.3	1	1.4	1
c-fat500-10	500	78,123	126	3.5	1	3.6	1
hamming8-2	256	1,024	128	0.1	1	0.1	1
hamming8-4	256	11,776	16	12.0	1,311	11.6	1,682
johnson8-2-4	28	168	4	<0.1	8	<0.1	8
johnson8-4-4	70	560	14	<0.1	9	<0.1	9
johnson16-2-4	120	1,680	8	49.7	153,168	79.4	183,305
keller4	171	5,100	11	5.1	5,060	6.7	6,217
p_hat300-1	300	33,917	8	1.6	883	1.7	885
p_hat300-2	300	22,922	25	45.9	3,684	43.6	7,488

We will compare Algorithm AB to Cliquer (see users.tkk.fi/~pat/cliquer.html), a freely available implementation of both Östergård's algorithm in [11] and an extension of it to the MWIS problem. Cliquer is faster than Algorithm AB on almost all the random graphs we have encountered so far. But there are other classes of randomly-generated graphs for which this is not the case.

We developed a random graph-generation process based on degree sequences

Table XI. Algorithm AB on weighted random graphs

			AB	
$\|V\|$	p	$\|E\|$	Time	Nodes
100	0.5	2,428–2,540	0.1	68.8
100	0.4	1,928–2,053	0.2	125.2
100	0.3	1,445–1,524	0.4	290.3
100	0.2	937–1,049	1.0	755.3
100	0.1	455–518	1.6	1,331.2
200	0.5	9,851–10,052	3.0	1,018.7
200	0.4	7,839–8,095	10.4	2,823.8
200	0.3	5,887–6,055	41.0	12,158.8
200	0.2	3,939–4,057	300.1	121,749.5

First specify the vertex set and the degree of each vertex; the number of edges is thus predetermined. The ends of each edge must be selected, and they are selected randomly whenever possible. When a random selection is possible, randomly choose a vertex currently adjacent to fewer vertices than its desired degree, randomly choose another such vertex to which the first is not already adjacent, and create an edge between the two vertices. A random selection is not possible if at any point a vertex v requires r additional neighbors to achieve its desired degree, and yet only r other vertices not already adjacent to v require any additional neighbors at all. In that case, make v adjacent to all r of those vertices.

We compared Algorithm AB and Cliquer on graphs with several types of degree sequences. An elementary choice is to make all vertex degrees the same, thus producing a regular graph. For several classes of sparse regular graphs, Algorithm AB beats Cliquer handily. For others, the opposite is true. Table XII gives some examples. Each line of the table gives average run times across ten graphs with the given parameters. All the graphs have vertex weights drawn from DiscreteU$[1, 10]$.

Table XIII gives results comparing these two algorithms on graphs whose vertices have one of two degrees. For each pair a, b under the heading "Degree",

Table XII. Algorithm AB and Cliquer on regular graphs

$\|V\|$	Degree	AB Time	Cliquer Time
100	3	0.4	77.2
100	4	0.9	14.2
100	5	2.0	9.0
100	6	2.6	3.1
100	7	3.5	1.8
100	8	2.7	1.1
100	9	2.5	0.9
100	10	2.3	0.3
120	3	1.5	1,953.9
120	4	4.3	514.0
120	5	14.7	267.0
120	6	16.5	116.3
120	7	20.6	48.9
120	8	18.2	26.2
120	9	24.2	15.3
120	10	23.5	7.6
140	3	2.5	>10,000
140	4	30.9	>10,000
140	5	71.1	9,754.2
140	6	70.0	4,655.5
140	7	161.0	946.9
140	8	185.6	527.0
140	9	226.7	262.6
140	10	115.7	191.5

the graphs generated have 80 vertices with degree a and 20 vertices with degree b. Again, all vertex weights are drawn from DiscreteU$[1, 10]$, and the results on each line are averages across ten graphs.

We investigated the use of Algorithm AB as a sub-problem solver for the algorithm of Warrier *et al.* described in Chapter II. The best results reported in [16] use (a weighted analog of) the Carraghan-Pardalos Algorithm as the sub-problem solver and partition the vertex set using METIS. For our investigation, we used the same METIS-based partition and decomposition, but we substituted Algorithm AB for the Carraghan-Pardalos Algorithm.

We obtained an implementation of the algorithm described in [16]. This implementation was not the same used to produce the results in [16], so we solved each problem under consideration using both sub-problem solvers in order to ob-

Table XIII. Algorithm AB and Cliquer on random graphs with two distinct vertex degrees

		AB	Cliquer
$\|V\|$	Degrees	Time	Time
100	5,20	3.4	>1,000
100	5,25	0.8	438.5
100	5,30	0.5	171.0
100	5,35	0.2	8.7
100	5,40	0.1	0.3
100	10,20	>1,000	298.9
100	10,25	388.6	105.7
100	10,30	161.2	73.0
100	10,35	183.2	107.6
100	10,40	38.2	90.2

tain comparable run times. We solved these problems using the same equipment we used to produce the other results in this chapter.

The results of these tests are presented in Tables XIV and XV. In both tables, the columns headed by "CP" indicate the performance using the Carraghan-Pardalos Algorithm as the sub-problem solver, and the columns headed by "AB" indicate the results when Algorithm AB is the sub-problem solver.

Table XIV compares the performances of the two sub-problem solvers on DIMACS graphs. The number of branch-and-bound nodes required in each case is similar, as we should expect; the number of nodes can vary from one technique to the other only if alternate LP optimizers result in different branching activities. The time required using Algorithm AB as the sub-problem solver is less than that required by using the CP Algorithm in each case listed.

Table XV compares the performances of the two sub-problem solvers on random graphs. These random graphs are the same ones used to test the algorithm of Warrier *et al*. in [16]. Each line of the table gives the average results on five graphs having the same maximum weight and the same edge probability. As with the DIMACS graphs, the number of branch-and-bound nodes required in

Table XIV. Two sub-problem solvers used within the algorithm of Warrier *et al*. on DIMACS graphs

					CP		AB	
Graph	$\|V\|$	$\|E\|$	$\alpha(G)$	$\|\hat{E}\|$	Time	Nodes	Time	Nodes
c-fat200-1	200	18,366	12	16,696	17.6	5	10.2	5
c-fat200-2	200	16,575	24	15,912	16.7	15	10.3	15
c-fat200-5	200	8,473	58	11,201	37.2	32	15.8	32
hamming8-2	256	1,024	128	846	0.8	1	0.2	1
hamming8-4	256	11,776	16	10,650	12.1	1	9.1	1
johnson8-2-4	28	168	4	113	0.2	7	0.1	7
johnson8-4-4	70	560	14	433	1.2	1	0.1	1
johnson16-2-4	120	1,680	8	1,243	1.1	1	0.8	1
keller4	171	5,100	11	4,499	2,411.7	14,123	1,247.6	14,082
p_hat300-1	300	33,917	8	31,511	873.1	1,308	511.9	1,360

each case is similar. The time required, however, is less for Algorithm AB than for the Carraghan-Pardalos Algorithm in every instance, with the former exhibiting 200 times the speed of the latter in some cases.

H. Conclusions

We recognized that what we call the Chordal Method used in [5] to generate clique covers serves primarily as a heuristic in the overall algorithm. Its inferiority to what we call the Greedy Method of [5] was demonstrated by our experiments in Section D. The branching scheme in [5] and the use of clique covers to enforce the upper bound required by that branching scheme stand as the chief contributions of that paper. We found that adapting the Greedy Method to force heavy independent sets to be covered yields considerable computational savings at small additional computational cost per subproblem.

This change to the Greedy Method as implemented in Algorithm A performs better even than the Babel Algorithm in [27]. Better yet is our Algorithm B that produces the same weighted clique covers that the Babel Algorithm does but is more computationally efficient.

Better than either of these is our hybrid Algorithm AB. It combines the superiority of the Babel-style (Dsatur) weighted clique covers at the upper nodes of the branch-and-bound tree with the speed of Method A at the lower nodes. Algorithm AB outperforms the state-of-the-art algorithm Cliquer on some graphs. We also demonstrated that Algorithm AB is an apt replacement for the Carraghan-Pardalos Algorithm as a sub-problem solver for the decomposition approach of Warrier *et al.*.

Some avenues of research appear to have potential due to these results. Where other graph structures have been used in addition to weighted (or unweighted) clique covers to produce branch sets for use with the Balas-Yu Branching Scheme, one could seemingly apply our Methods A and B to obtain computational savings over the original implementations.

Method B first covers the vertices of a maximal independent set. Those vertices are chosen by GKD, with ties broken by weight. Perhaps a different tie-breaker that also takes into account vertex degree or the weight of non-neighbors would produce heavier initially covered independent sets, resulting in better weighted clique covers.

Algorithm AB is a very crude hybrid. It could be improved by employing a heuristic that gauges the difficulty of the sub-problem to be solved at the present branch-and-bound node and applies Method B only if the problem seems difficult enough to benefit from its higher-quality but more expensive weighted clique covers.

Given that Algorithm AB's performance relative to that of Cliquer is sensitive to graph structure, it seems profitable to test these algorithms on other varieties of graphs.

Table XV. Two sub-problem solvers used within the algorithm of Warrier *et al*. on 100-vertex random graphs

		CP		AB	
Max Wt	p	Time	Nodes	Time	Nodes
1	0.50	51.4	718.2	17.1	645.0
1	0.40	105.8	1,339.4	37.5	1,320.8
1	0.30	191.9	1,826.8	50.2	1,982.2
1	0.20	679.1	3,104.2	112.1	3,237.6
1	0.15	1,160.0	2,279.0	87.8	2,040.8
1	0.10	1,532.9	766.6	41.4	741.2
1	0.05	310.5	14.2	1.9	17.0
1	0.01	19.3	1.0	0.1	1.0
20	0.50	27.7	339.4	10.1	387.0
20	0.40	46.0	482.6	14.5	472.8
20	0.30	51.7	331.6	9.5	296.6
20	0.20	164.2	414.2	17.7	435.0
20	0.15	259.1	355.6	16.7	350.2
20	0.10	156.0	25.0	2.3	24.8
20	0.05	136.2	7.8	0.6	7.2
20	0.01	7.2	1.0	0.1	1.0
40	0.50	27.7	339.4	10.1	387.0
40	0.40	46.0	482.6	14.5	472.8
40	0.30	51.7	331.6	9.5	296.6
40	0.20	164.2	414.2	17.7	435.0
40	0.15	259.1	355.6	16.7	350.2
40	0.10	156.0	25.0	2.3	24.8
40	0.05	136.2	7.8	0.6	7.2
40	0.01	7.2	1.0	0.1	1.0
60	0.50	21.6	264.8	6.5	248.8
60	0.40	37.4	375.2	12.0	381.8
60	0.30	82.3	592.0	19.8	608.6
60	0.20	165.5	385.6	14.5	344.0
60	0.15	180.5	188.6	9.4	190.4
60	0.10	286.5	72.8	5.5	71.6
60	0.05	42.8	1.2	0.3	1.2
60	0.01	5.8	1.0	0.1	1.0
100	0.50	25.8	325.6	9.8	369.8
100	0.40	46.7	510.8	16.1	562.0
100	0.30	57.5	369.4	12.5	390.0
100	0.20	178.8	525.4	19.7	530.6
100	0.15	358.6	544.0	24.3	539.0
100	0.10	560.9	164.6	11.1	161.8
100	0.05	142.6	7.6	0.6	7.0
100	0.01	7.0	1.0	0.1	1.0

CHAPTER IV

ODD-HOLE-PRESERVING GRAPH REDUCTIONS

A. Introduction

We investigate in Section B some structures in graphs that are related to the presence of holes in graphs. Section C discusses techniques by which subgraphs of a given graph can be found such that the subgraphs preserve the presence of odd holes in the given graph. Section D explores properties of such subgraphs that can simplify the search for odd holes in them.

B. Structures Related to the Presence of Holes

A graph is **biconnected** if between every two vertices there exist two independent paths. A graph G must have a biconnected subgraph to contain a hole (or even a cycle), and every hole of G belongs to some biconnected subgraph of G.

A **clique** of G is a set $K \subseteq V(G)$ such that the vertices in K are pair-wise adjacent. A **vertex separator** of a connected graph G is a set $D \subseteq V(G)$ such that $G - D$ is not connected. If G has a separator K that is also a clique, and $G - K$ has components $C_1, \ldots, C_k$, then no hole in G contains vertices from more than one C_i. Further, any hole in G is a hole in exactly one $G[C_i \cup N(C_i)]$.

Two distinct vertices u and v of G are **twins** if $N(u) = N(v)$. For twins u and v and a hole H containing u but not v, $(H \setminus \{u\}) \cup \{v\}$ is a hole of G. For a different hole H containing both u and v, we can easily see that $|H| = 4$, since u and v must each have two neighbors in $G[H]$, and these neighbors must also be neighbors of both u and v in G.

Two adjacent vertices u and v of G are **pseudo-twins** if $N(u) \triangle N(v) = \{u, v\}$

Pseudo-twins u and v belong to no common hole, for they induce a triangle with any one of their common neighbors. As with twins, for a hole H containing u but not v, $H \setminus \{u\} \cup \{v\}$ is a hole of G.

For a hole H in G and a vertex $v \in H$, $\bar{N}(v) \cap H$ induces an odd-length path in G. Therefore, no vertex v in G belongs to a hole in G if $\bar{N}(v)$ induces no edge.

A graph is **bipartite** if the vertex set can be partitioned into two independent sets. Bipartite graphs cannot contain odd cycles, let alone odd holes.

C. Hole-Preserving Graph Reductions

While the above structural results are simple and well-known, their systematic use to simplify the odd hole detection problem is not reported in the literature. While previous work has used results on sufficient conditions for the presence of odd holes (concerning jewels, proper wheels, pyramids, etc. — see [18, 31]), the results in Section B suggest some necessary conditions that have not been investigated. We determine in this section a constructive approach to using these results as necessary conditions.

We call a graph **reduced** if it

1. is biconnected,
2. has no clique separator,
3. has no twins or pseudo-twins,
4. has no vertices whose non-neighbors induce no edge, and
5. is not bipartite.

The following steps for a graph G are called the **reduction process**. During the reduction process, we create a collection $\mathcal{R}$ of reduced subgraphs of G. We draw these from $\mathcal{Q}$, a collection of subgraphs whose status as reduced or non-reduced has not yet been determined.

1. Add G to $\mathcal{Q}$.
2. Remove an arbitrary subgraph F from $\mathcal{Q}$.
3. If F is not biconnected, then find its maximal two-connected subgraphs $B_1, \ldots, B_k$, and add each graph $F[B_i]$ to $\mathcal{Q}$. Go to Step 2.
4. If F has a clique separator whose deletion yields components $C_1, \ldots, C_k$, then add each graph $F[C_i \cup N(C_i)]$ to $\mathcal{Q}$. Go to Step 2.
5. If F has twins or pseudo-twins u and v, then add $F - v$ to $\mathcal{Q}$. Go to Step 2.
6. If F has a vertex v such that $F[\bar{N}(v)]$ has no edges, then add $F - v$ to $\mathcal{Q}$. Go to Step 2.
7. If F is bipartite, then go to Step 2.
8. Add F to $\mathcal{R}$.
9. If $\mathcal{Q}$ is not empty, go to Step 2.

If G has an odd hole on $2k + 1$ vertices, then, by the properties explored in Section B, some graph in $\mathcal{R}$ will have an odd hole of that order after applying the reduction process to G.

We can find the maximal two-connected subgraphs of G in $O(|V| + |E|)$ time by an algorithm of Tarjan [32]. We can find all clique separators of G in $O(|V|+|E|)$ time by another algorithm of Tarjan [33]. We can determine if u and v are twins or pseudo-twins in $O(|E|)$ time. We can determine if a vertex v of $G = (V, E)$ has non-neighbors that induce an edge in $O(|E|)$ time. We can determine if $G =$

(V, E) is bipartite in $O(|V| + |E|)$ time. So each step of the reduction process takes polynomial time.

The total number of graphs in $\mathcal{Q}$ and $\mathcal{R}$ increases only when Steps 3 and 4 are successfully applied. In all other cases, a graph in $\mathcal{Q}$ is transferred to $\mathcal{R}$, a single graph in $\mathcal{Q}$ is replaced by another graph, or a single graph in $\mathcal{Q}$ is discarded. At each successful application of Step 3 with $k \geq 2$, at least $\binom{k}{2}$ edges of $\bar{H}$ are no longer represented in any graph $H[B_i]$. Likewise, at each successful application of Step 4, at least $\binom{k}{2}$ edges of $\bar{H}$ are no longer represented in any graph $H[C_i \cup N(C_i)]$. As a result, the number of subgraphs produced by the end of the reduction procedure and the number of steps required to produce them are not more than $\binom{|V|}{2} - |E|$ (noted originally by Gavril in [34] with regard to clique separators). Thus, the reduction process requires polynomial time to complete. Its worst-case performance is $O(|V|^3 + |E|\,|V|^2)$.

D. Results for Reduced Graphs

The following results about cycles and holes hold for reduced graphs, although several of these permit weaker hypotheses.

1. Every reduced graph has an odd cycle of at least five vertices.
2. Every vertex in a reduced graph lies on an odd cycle.
3. Every vertex in a reduced graph lies on an odd cycle of at least five vertices.
4. Every edge of a reduced graph belongs to an odd cycle of at least five vertices.
5. Every vertex of a reduced graph lies on a hole.

We prove the theorems below. The first will be proved with both a stronger conclusion and weaker hypotheses than given above.

Theorem 1 *Any graph satisfying Parts 1, 2, and 5 of the definition of reduced graph has an odd cycle of at least five vertices. Further, every triangle of such a graph contributes at least two vertices to some odd cycle of at least five vertices.*

Proof. Let G be reduced. Since it is non-bipartite, it has an odd cycle C. If $|C| \geq 5$, then we are done. If not, $C = \{a, b, c\}$ is a triangle, and we may construct a larger odd cycle as follows.

Since C is a clique, but G is reduced, there exists a vertex $d \in \bar{N}(a) \cup \bar{N}(b) \cup \bar{N}(c)$. G is biconnected, so there exist two d-C paths P and Q having only d in common. Without loss of generality, we assume that P and Q end in C at a and b, respectively. If the length of the path $aPdQb$ is $2k$ for some positive integer $k \geq 2$, then $aPdQba$ is a cycle of odd length $2k + 1 \geq 5$. If its length is $2k + 1$ for some positive integer k, then $aPdQbca$ is a cycle of odd length $2k + 3 \geq 5$. Otherwise, $aPdQb$ has length two; P and Q are the edges ad and bd, respectively. But d cannot be a neighbor of all three vertices a, b, and c, so $d \in \bar{N}(c)$.

G is reduced, so $\left|\bar{N}(a)\right| \geq 2$ because $\bar{N}(a)$ induces an edge. But $\{b, c, d\} \subseteq N(a)$, so there exists a vertex $e \in \bar{N}(a)$ distinct from all vertices that we have identified so far. Since G is biconnected, there exist two e-$\{a, b, c, d\}$ paths, R and S, having only e in common. R and S have ends in $\{a, b, c, d\}$; call them p and q, respectively. If the sum of the lengths of R and S is odd, and thereby at least three, then a path T of length two in $G[a, b, c, d]$ from p to q would suffice to produce the cycle we want, for the length of $eRpTqSe$ would be odd and at least five. But inspection reveals paths of length two in $G[a, b, c, d]$ between all pairs of its vertices.

Suppose now that the sum of the lengths of R and S is even, hence at least two. A path T of length three in $G[a, b, c, d]$ would suffice to produce the cycle we want, for the length of $eRpTqSe$ would be odd and at least five. But inspection reveals length-three (Hamiltonian) paths in $G[a, b, c, d]$ having any pair of ends except a and b.

So, suppose that $\{p, q\} = \{a, b\}$; without loss of generality, $a = p$ and $b = q$. Since e is not adjacent to a, the length of R is at least two, whereby the sum of the lengths of R and S is at least four (since it is even and at least three). The length of the cycle $eRabSe$ is odd and at least five.

The second part of our conclusion follows by noting that after we supposed above that C was a triangle, every cycle that we constructed to satisfy the first part of the conclusion included at least two vertices of C. □

We will prove a stronger version of the second theorem listed above. Parts 2, 3, and 4 of the definition of reduced graph need not hold for the result to follow.

Theorem 2 *Every vertex in a biconnected non-bipartite graph belongs to an odd cycle.*

Proof. Let G be biconnected and non-bipartite. Since G is non-bipartite, it has an odd cycle C. If C is G, then the theorem holds; otherwise, let $v \in V(G) \setminus C$. Since G is biconnected, there exist v-C paths P and Q having only v in common. Let P and Q end in p and q on C. For some paths R and S, C is $pRqSp$. Without loss of generality, take the length of R to be even and the length of S to be odd. One of $vPpRqQv$ and $vPpSqQv$ is an odd cycle. □

Theorem 3 *Let G be reduced, and let C be an odd cycle in G of at least five vertices. Then every vertex of G lies on an odd cycle of at least five vertices that includes at least* *three vertices of C.*

Proof. The conclusion holds for all vertices in C, so pick $v \in V(G) \setminus C$. Since G is biconnected, there exist two v-C paths having only v in common. Choose two such paths, P and Q, such that the sum of their lengths is maximum among all such pairs of paths. P and Q end in distinct vertices p and q, respectively, on C. For some paths R and S, C is $pRqSp$. Without loss of generality, take the length of R to be even and the length of S to be odd. If the sum of the lengths of P and Q is odd (hence at least three), then the length of the cycle $vPpRqQv$ is odd and at least five. If their sum is even, then the length of the cycle $vPpSqQv$ is odd, and that length is less than five (*i.e.*, is three) only if P, Q, and S have length one.

Suppose this is the case, and note that p and q are consecutive on C, hence adjacent. Let $C' = C \setminus \{p, q\}$. Since G has no clique separator, $\{p, q\}$ does not separate v from C', and there exists a v-C' path R not including p and q. Its end in C' we name r. R is also a v-C path. P and R are independent v-C paths, as are Q and R, so the maximality of the choice of P and Q requires that R be the edge vr.

Without loss of generality, r is nearest q on C. Let S and T be paths such that C is $pqSrTp$. If the length of S is even, then the length of cycle $vrSqpv$ is odd and at least five. If the length of S is odd and at least three, then the length of T is at least three by the minimality of S and is odd because of the odd number of vertices in C. If the length of S is one, then the length of T is odd and at least three because the number of vertices in C is odd and at least five. So the length of the cycle $vrTpv$ is odd and at least five.

The second part of our conclusion follows by noting that every cycle that we constructed to satisfy the first part of the conclusion included at least three vertices of C. $\square$

Theorem 4 *Every edge of a reduced graph belongs to an odd cycle of at least five vertices.*

Proof. Let G be reduced, and let $u, v \in V$ be adjacent. By Theorem 3 there exists an odd cycle C of at least five vertices in G that contains u. If $v \in V(C)$ and u and v are consecutive on C, then we are done. Suppose $v \in V(C)$ and u and v are not consecutive on C. There exists a u-v path P in C of length $2k$ for some positive integer k. The cycle $uvPu$ has odd length $2k + 1$. If $k \geq 2$, then we are done, so suppose $k = 1$. Then P is a path uwv for some $w \in V(C)$.

Let $C' = C - \{u, v, w\}$. G has no clique separators, so there exists a w-C' path Q in $G - \{u, v\}$ having end x in C'. There are paths E and O having even and odd lengths, respectively, such that C is $uwvExOu$, the placements of u and v being made without loss of generality. Let the length of E be $2m$ for some positive integer m, and let the length of O be $2n + 1$ for some non-negative integer n. If Q has odd length $2r + 1$ for some non-negative integer r, then the cycle $vuwQxEv$ has length

$$1 + 1 + (2r + 1) + 2m = 2(m + r) + 3,$$

which is odd and at least five. If Q has even length $2s$ for some positive integer s, then the cycle $uvwQxOu$ has length

$$1 + 1 + 2s + (2n + 1) = 2(n + s) + 3,$$

which is odd and at least 5.

So assume $v \notin V(C)$. If v has a neighbor w in $V(C)$ such that u and w are not consecutive on C, then C contains a u-w path O of odd length $2m + 1$ for some positive integer m, and the cycle $uvwOu$ has length

$$1 + 1 + (2m + 1) = 2m + 3,$$

which is odd and at least five. So suppose v has no such neighbor in $V(C)$.

G is biconnected, so there exists a v-$(C - u)$ path in $G - u$; choose a longest such path P, and let its end in $C - u$ be named w. If P has even length $2k$ for some positive integer k, then we note that C contains a u-w path E of even length $2m$ for some positive integer m, and the cycle $uvPwEu$ has length

$$1 + 2k + 2m = 2(k + m) + 1,$$

which is odd and at least five.

So suppose P has odd length $2k + 1$ for some non-negative integer k. If k is positive, then we note that C contains a u-w path O of odd length $2m + 1$ for some non-negative integer m, and the cycle $uvPwOu$ has length

$$1 + (2k + 1) + (2m + 1) = 2(k + m) + 3,$$

which is odd and at least five.

So suppose that $k = 0$, whereby P is the edge vw. Since $w \in V(C)$, u and w are consecutive on C, and are thus adjacent in both C and G. Let $C' = C - \{u, w\}$. G has no clique separator, so there exists a v-C' path Q ending in some x in $V(C')$. Q is also a v-C path, so its length is one by the maximality of P, whereby Q is the edge $\{v, x\}$. By our assumption, u and x are consecutive on C, so v has no other neighbors in $V(C)$.

Suppose v has a neighbor v' in $V(G-C)$. Since G is biconnected, there exists a v'-C path P' in $G-v$. But then $vv'P'$ is a v-C path longer than P, a contradiction. So $N(v) = \{u, w, x\}$. G has no pseudo-twins, so $N(u) \triangle N(v) \neq \{u, v\}$, whereby u has some neighbor y distinct from v, w, and x.

If $y \in V(C)$, then we note that it is a vertex of $C-u$, which is a w-x path that has odd length at least three. Without loss of generality, y has an even distance from w and an odd distance from x in $C-u$. So $C-u$ contains a unique y-w path E having even length $2m$ for some positive integer m, and the cycle $uvwEyu$ has length

$$1+1+2m+1=2m+3,$$

which is odd and at least five.

So suppose $y \notin V(C)$. G is biconnected, so there exists a y-$(C-u)$ path Q in $G-u$ with an end z in $C-u$. Note that v is not a vertex of Q.

Note that z is a vertex of $C-u$. Without loss of generality, z has an odd distance from w and an even distance (possibly zero) from x in $C-u$. So $C-u$ is $wOzEx$ for two unique paths E and O having lengths $2m$ and $2n+1$, respectively, for some non-negative integers m and n (at least one of which is positive).

If Q has odd length $2k+1$ for some non-negative integer k, then the cycle $uvwOzQyu$ has length

$$1+1+(2n+1)+(2k+1)+1=2(n+k)+5,$$

which is odd and at least five. If Q has even length $2k$ for some positive integer k, then the cycle $uvxEzQyu$ has length

$$1+1+2m+2k+1=2(m+k)+3,$$

which is odd and at least five. □

Theorem 5 *Every vertex of a reduced graph lies on a hole.*

Proof. Let G be reduced, and let $v \in V(G)$. $\bar{N}(v)$ induces an edge, so it has a component D that induces an edge. Let

$$N(D) = \left(\bigcup_{d \in D} N(d) \right) \setminus D,$$

noting that $N(D) \subseteq N(v)$: clearly $v \notin N(D)$, and having a nonempty $\bar{N}(v) \cap N(D)$ contradicts the choice of D as a component of $\bar{N}(v)$.

$N(D)$ separates v and D, so $N(D)$ is not a clique. Choose two non-adjacent vertices $w, x \in N(D)$. $G[D]$ is connected, and both w and x have neighbors in D, so $G[D \cup \{w, x\}]$ is connected. So there exists a shortest x-w path P in $G[D \cup \{w, x\}]$. Since v has no neighbors in D and hence among the internal vertices of P, $vwPxv$ is a hole in G. $\square$

E. Conclusions

We have explained a method by which a graph may be reduced in polynomial time to a polynomial number of subgraphs such that, if the original graph contains an odd hole, one of the provided subgraphs contains an odd hole. Thus the odd-hole detection problem can be decided in polynomial time if and only if it can be decided in polynomial time on reduced subgraphs.

We have also provided promising results on the abundance of holes and large odd cycles in such reduced graphs. Further, the proof techniques that we have demonstrated address the construction of paths and cycles in reduced graphs at a fundamental level and could be useful in developing new structural results for reduced graphs.

CHAPTER V

CONCLUSIONS

A. Independent Set Problems

The decomposition approach to the MWIS problem of Warrier *et al*. in [16] is a novel method. Chapter II demonstrates that it has been advanced by our initialization and RMP-tightening techniques. It can also make use of our partitioning method that produces chordal subgraphs.

It may be possible to find a partitioning method for this decomposition approach that performs better than METIS. It would almost certainly need to combine small values of $\left|\hat{E}\right|$ with easily solved sub-problems.

In Chapter III, we recognized that the Chordal Method used in [5] by Balas and Yu is inferior to their Greedy Method. Their branching scheme, however, has proved very useful in other work. We found that adapting the Greedy Method to force heavy independent sets to be covered (as does the Babel Algorithm) yields considerable computational savings at small additional computational cost per subproblem. The resulting Algorithm A performs better even than the Babel Algorithm. Better yet is our Algorithm B that produces the same weighted clique covers that the Babel Algorithm does but is more computationally efficient. Our hybrid Algorithm AB outperforms the state-of-the-art algorithm Cliquer on a few graphs. It is also an apt replacement for the Carraghan-Pardalos Algorithm as a sub-problem solver for the decomposition approach of Warrier *et al*..

Method B, our modification of Dsatur, might be improved by a suitable replacement for GKD, especially one that imitates the order imposed by GKD when a tiebreaker is not called for.

Algorithm AB could be improved by employing a heuristic that assesses whether Method A or Method B is appropriate for solving a particular subproblem. Given that Algorithm AB's performance relative to that of Cliquer is sensitive to graph structure, it seems worthwhile to test both the Babel Algorithm and Algorithm AB on other varieties of graphs.

B. Odd-Hole-Preserving Graph Reductions

In Chapter IV, we developed an odd-hole reduction process. By applying it, a graph is reduced in polynomial time to a polynomial number of its subgraphs such that, if the original graph contains an odd hole, one of the provided subgraphs contains an odd hole. Thus the odd-hole detection problem can be decided in polynomial time if and only if it can be decided in polynomial time on reduced subgraphs.

We have also provided promising results on the abundance of holes and large odd cycles in such reduced graphs. Further, the proof techniques that we have demonstrated address the construction of paths and cycles in reduced graphs at a fundamental level and could be useful in developing new structural results for reduced graphs.

REFERENCES

[1] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, New York: W.H. Freeman and Company, 1979.

[2] I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of Combinatorial Optimization*, D.Z. Du and P.M. Pardalos, Eds., pp. 1–74. Boston: Kluwer Academic Publishers, 1999.

[3] A. Mehrotra and M.A. Trick, "A column generation approach for graph coloring," *INFORMS Journal on Computing*, vol. 8, no. 4, pp. 344–354, 1996.

[4] C. Bron and J. Kerbosch, "Finding all cliques in an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, September 1973.

[5] E. Balas and C.S. Yu, "Finding a maximum clique in an arbitrary graph," *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1054–1068, November 1986.

[6] E. Balas and J. Xue, "Minimum weighted coloring of triangulated graphs, with application to maximum weight vertex packing and clique finding in arbitrary graphs," *SIAM Journal on Computing*, vol. 20, no. 2, pp. 209–221, April 1991.

[7] E. Balas and J. Xue, "Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring," *Algorithmica*, vol. 15, pp. 397–412, 1996.

[8] J.-M. Bourjolly, G. Laporte, and H. Mercure, "A combinatorial column generation algorithm for the maximum stable set problem," *Operations Research Letters*, vol. 20, pp. 21–29, 1997.

[9] R. Carraghan and P.M. Pardalos, “An exact algorithm for the maximum clique problem,” *Operations Research Letters*, vol. 9, no. 6, pp. 375–382, November 1990.

[10] C. Mannino and A. Sassano, “Edge projection and the maximum cardinality stable set problem,” in *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, D.S. Johnson and M.A. Trick, Eds., vol. 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[11] P.R.J. Östergård, “A fast algorithm for the maximum clique problem,” *Discrete Applied Mathematics*, vol. 120, pp. 197–207, 2002.

[12] E.C. Sewell, “A branch and bound algorithm for the stability number of a sparse graph,” *INFORMS Journal on Computing*, vol. 10, no. 4, pp. 438–447, 1998.

[13] D.R. Wood, “An algorithm for finding a maximum clique in a graph,” *Operations Research Letters*, vol. 21, pp. 211–217, 1997.

[14] E.R. Barnes, “A branch-and-bound procedure for the largest clique in a graph,” in *Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems*, P.M. Pardalos, Ed., pp. 63–77. Boston: Kluwer Academic Publishers, 2000.

[15] F. Rossi and S. Smriglio, “A branch-and-cut algorithm for the maximum cardinality stable set problem,” *Operations Research Letters*, vol. 28, pp. 63–74, 2001.

[16] D. Warrier, W.E. Wilhelm, J.S. Warren, and I.V. Hicks, “A branch-and-price

approach for the maximum weight independent set problem," *Networks*, vol. 46, no. 4, pp. 198–209, 2005.

[17] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas, "Progress on perfect graphs," *Mathematical Programming Series B*, vol. 97, pp. 405–422, 2003.

[18] M. Chudnovsky and P. Seymour, "Recognizing Berge graphs," January 2003, manuscript, available at citeseer.ist.psu.edu/chudnovsky03recognizing.html.

[19] D. Bienstock, "On the complexity of testing for odd holes and induced odd paths," *Discrete Mathematics*, vol. 90, pp. 85–92, 1991.

[20] M. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, New York: Academic Press, 1980.

[21] A. Frank, "Some polynomial algorithms for certain graphs and hypergraphs," in *Proceedings of the 5th British Combinatorial Conference*, C. St. J. A. Nash-Williams and J. Sheehan, Eds., Winnipeg, 1975, vol. 15 of *Congressum Numerantium*, pp. 211–226, Utilitas Mathematica.

[22] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, pp. 359–392, 1998.

[23] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," Tech. Rep. 98-019, Army HPC Research Center, Department of Computer Science, University of Minnesota, 1998.

[24] G. Karypis and V. Kumar, "Multilevel *k*-way partitioning scheme for irregular graphs," *J. Parallel Distrib. Comput.*, vol. 48, pp. 96–129, 1998.

[25] D.S. Johnson and M.A. Trick, Eds., *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, vol. 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, 1996.

[26] A.M. Geoffrion, "Lagrangean relaxation for integer programming," *Mathematical Programming Study*, vol. 2, pp. 82–114, 1974.

[27] L. Babel, "A fast algorithm for the maximum weight clique problem," *Computing*, vol. 52, pp. 31–38, 1994.

[28] D.B. West, *Introduction to Graph Theory*, Englewood Cliffs, NJ: Prentice Hall, second edition, 2001.

[29] D.J. Rose, R.E. Tarjan, and G.S. Lueker, "Algorithmic aspects of vertex elimination on graphs," *SIAM Journal on Computing*, vol. 5, pp. 266–283, 1976.

[30] D. Brélaz, "New methods to color the vertices of a graph," *Comm. of the ACM*, vol. 22, no. 4, pp. 251–256, April 1979.

[31] M. Conforti, G. Cornuéjols, and G. Zambelli, "Decomposing Berge graphs containing no proper wheel, long prism or their complements," *Combinatorica*, vol. 26, no. 5, pp. 533–558, 2006.

[32] R.E. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, pp. 146–160, 1972.

[33] R.E. Tarjan, "Decomposition by clique separators," *Disc. Math.*, vol. 55, pp. 221–232, 1985.

[34] F. Gavril, "Algorithms on clique separable graphs," *Disc. Math*, vol. 19, pp. 159–165, 1977.

VITA

Name:	Jeffrey S. Warren
Address:	Department of Industrial and Systems Engineering
	Texas A&M University, College Station, Texas 77843-3131
E-mail Address:	j-warren@tamu.edu
Education:	B.S., Mathematics, Abilene Christian University, 1996
	M.S., Mathematics, Texas A&M University, 1999