Combinatorial Branch-and-Bound for the Maximum Weight Independent Set Problem

Jeffrey S. Warren and Illya V. Hicks
Department of Industrial and Systems Engineering, Texas A&M University
College Station, Texas 77840-3131, USA
[j-warren@tamu.edu, ivhicks@tamu.edu]

August 7, 2006

We describe three combinatorial branch-and-bound solvers for the maximum weight independent set problem. All use weighted clique covers to generate upper bounds, and all branch according to the method of Balas and Yu (1986). One extends and speeds up the method of Babel (1994). A second one employs a modified method from Balas and Yu (1986) to produce clique covers that share structural similarities with those produced by Babel (1994). 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

Keywords: maximum weight independent set, weighted clique cover, branch-and-bound

1. Introduction

For a graph G, $S \subseteq V(G)$ is an **independent set** if no vertices in S are adjacent. The **maximum independent set problem** is the problem of finding an independent set of largest cardinality in a given graph. The **maximum weight independent set problem** is the problem of finding an independent set of greatest total weight in a given graph according to some weighting of the vertices. Both problems are NP-hard (see Garey and Johnson, 1979). They have application in coding theory, combinatorial auctions, computer vision, and protein chemistry (see Bomze et al., 1999).

Many have worked to solve both problems exactly. Techniques include explicit enumeration of maximal independent sets (Bron and Kerbosch, 1973), combinatorial branch-and-bound (Balas and Xue, 1991, 1996; Balas and Yu, 1986; Bourjolly et al., 1997; Carraghan and Pardalos, 1990; Mannino and Sassano, 1996; Östergård, 2002; Sewell, 1998; Wood, 1997), and continuous formulations under branch-and-bound (Barnes, 2000; Mehrotra and Trick, 1996).

We refer the reader to Bomze et al. (1999) for a discussion of these lines of research.

Clique covers have been used to provide upper bounds in successful branch-and-bound solvers for both problems. We describe herein two branch-and-bound algorithms based on clique covers. The first uses a cover-creation routine that is procedurally like those in Balas and Yu (1986) and Balas and Xue (1991), but whose clique covers borrow structural features from those created by the algorithm in Babel (1994). The second is a more direct improvement on Babel's algorithm that significantly reduces the run time required on integer-weighted graphs (which Babel's algorithm handles) and generalizes to the case of real-weighted graphs. The two algorithms yield 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 3 summarizes these previous studies, and Section 4 contains our analysis of that strand of research, including some experimental work. Sections 5 and 6 describe our contributions that arose from that analysis, and Section 7 relates our subsequent computational results. Our conclusions from these new computations are given in Section 8.

First we will address some preliminary matters.

2. Preliminaries

For basic terminology and an introduction to graph theory, we refer the reader to West (2001). All graphs in this paper are finite and simple. We assume in this paper that the vertex set and edge set 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 $w : V \to \mathbb{R}$; 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 $W(\emptyset) = 0$.

An **independent set** of graph G is any set $S \subseteq V$ such that no vertices in S are adjacent. The **maximum independent set problem** is to find an independent set S of G such that $|S|$ is maximum among all independent sets of G. This maximum is denoted $\alpha(G)$. The **maximum weight independent set problem** is to find an independent set S of G such that $w(S)$ is maximum among all independent sets of G. This maximum is denoted $\alpha_w(G)$. Both problems are NP-hard (see Garey and Johnson, 1979).

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 Bomze et al. (1999), 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, 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:

$$\begin{aligned} \alpha(G) \leq \phi(G) &\leq t \\ \alpha_w(G) \leq \phi_w(G) &\leq \sum_{i=1}^{s} W_i. \end{aligned}$$

Analogous inequalities hold for colorings and clique numbers

3. Previous Work

3.1. 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 maximum and maximum weight independent set problems on them.

A graph is **chordal** if it induces no cycles other than triangles. 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 Rose et al. (1976), Balas and Yu (1986) developed 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 maximum independent sets 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 maximum independent set S_i in $G[V_i]$.

Either S or one of $S_1 \cup \{x_1\}, \ldots, S_k \cup \{x_k\}$ is a maximum independent set for G. The same method, of course, can be applied to the problem on each $G[V_i]$. Repeated application will yield maximum independent set 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 maximum independent set 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**. This algorithm corresponds to the variant called TC4 in Balas and Yu (1986). The root node of the branch-and-bound tree corresponds to the maximum independent set problem on G itself. Among all unsolved maximum independent set 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 Balas and Yu (1986) demonstrate that the Balas-Yu Algorithm handily outperforms its ablest predecessor, the algorithm in Bron and Kerbosch (1973).

3.2. Balas and Xue

Balas and Xue (1991) 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 $w : V \to \mathbb{R}$. By extending the coloring algorithm in Balas and Yu (1986), 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 Balas and Yu (1986) 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 Balas and Yu (1986) 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 plain, 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 maximum weight independent set 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 maximum weight independent set 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 (1986), Balas and Xue (1991) 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 paper, 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 $U_i = \{v \in V_i : w_{i+1}(v) = 0\}$, 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, 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 Balas and Xue (1991), 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 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 5.1).

When combined with a problem-selection and fathoming process analogous to that in Balas and Yu (1986), these weighted clique cover methods yield the **Balas-Xue Algorithm**. The computational results in Balas and Xue (1991) demonstrate that it outperforms its ablest predecessor, a weighted version of the algorithm in Carraghan and Pardalos (1990).

3.3. Babel

Babel (1994) 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 Brélaz (1979). The method in Babel (1994) for generating weighted clique covers, which we will refer to as the **Babel Method**, is likewise an analog to the coloring method in Brélaz (1979).

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 $D = \{v \in V : \mathcal{K}(v) \leq w(S)\}$, 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 (1994) reports 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 (1994) reported significant improvements over the performance of the Balas-Xue Algorithm in most cases, especially on low-density graphs.

4. 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 3.1). 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 the Chordal Method found at the root node a maximum independent set 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 were fortunate enough to find a maximum independent set. 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 a primarily heuristic function. If our assumptions in Section 3.2 about the Balas-Xue Algorithm are reasonable, then its Weighted Chordal Method would also seem to serve primarily as a heuristic. This is suggested by some of the results presented in Balas and Yu (1986, 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 3.2, whereby, for constant weight functions, our implementation reduces to an implementation of the Balas-Yu Algorithm) and two variants: one in which the Weighted Chordal Method is used at every node of the branch-and-bound tree and another in which the Weighted Greedy Method is likewise used. We call them the **Chordal Variant** and the **Greedy Variant**. We started the Greedy Variant by applying a modest greedy heuristic; see Section 6.1 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 maximum independent set and maximum weight independent set problems on several graph instances.

Table 1 shows results obtained by solving the maximum independent set problem on some graphs from the Second DIMACS Implementation Challenge (Johnson and Trick, 1996); the times are in seconds, and an asterisk (*) denotes instances in which computer memory was exhausted. Table 2 shows results obtained by solving the maximum weight independent set problem on some uniform random graphs with uniformly distributed vertex weights; see Section 7 for details about these graphs. Each line of Table 2 gives the average times and 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 1: 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 2: The Balas-Xue Algorithm and its Variants

			Balas-Xue		Greedy		Chordal	
$\lvert V\rvert$	p	$\lvert E\rvert$	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 (1986) generated only one random graph for each set of graph-generation parameters. Balas and Xue (1991) generated only two graphs per set, and Babel (1994) 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 Balas and Yu (1986)) and found it to be as good as or better than the Balas-Yu Algorithm for graphs with relatively small maximum independent sets but considerably worse for graphs with larger maximum independent sets. 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 2.

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 maximum independent set 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 maximum independent set S_T of $G[T]$, we are not guaranteed that $|I \cup S_T| \geq |S|$, even if S_T is also a maximum independent set 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 3 and 4. 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 3: The Modified Chordal Variant of the Balas-Yu Algorithm

Graph	$\lvert V\rvert$	$\lvert E\rvert$	$\alpha(G)$	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

Table 4: 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

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 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 7 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 maximum weight independent set 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.

5. New Clique Cover Methods

5.1. Method A

The Babel Method spends its first steps adding the vertices of a maximal independent set to its weighted clique cover. We find 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.

5.2. 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 vertex weights. 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 2. 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.

6. Implementation Issues

6.1. 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 7, we compare these algorithms with their respective forebears, the Balas-Xue Algorithm and the Babel Algorithm (but see the note on Algorithm A* in Section 7). 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 7 that it often outperforms both Algorithms A amd B. Since it is the best representative of our new work, we will compare it against the best general algorithm known for the maximum weight independent set problem, the algorithm of Östergård (2002). 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 (Carraghan and Pardalos, 1990; Sewell, 1998; Rossi and Smriglio, 2001), 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 3.3 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.

6.2. 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 4 and 5.1). This heuristic is applied to G to begin our Greedy Variant, as described in Section 4. 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 5.1).

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 choices 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 apply this initial heuristic to Algorithm B to allow a more direct comparison to the Babel Algorithm.

7. 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 U$[0, 1]$ and include the edge in the graph if $r < p$. Then for each vertex, generate a random deviate in 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 4). 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 see 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 5 and 6, respectively. In Table 6, 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 6 are the same as those of Tables 2 and 4 in Section 4.

Table 5: 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

Table 5 demonstrates that Algorithm A* outperforms Algorithm A only on johnson16-2-4 among the listed DIMACS graphs. In Table 6, 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 7. (Note that their performance is almost identical for unweighted graphs, so their results on the DIMACS graphs are reported further below in Table 9, where we compare them to the results of Algorithm AB.)

Table 6: 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 7: Algorithm B and the Babel Algorithm on weighted random graphs

			B	Babel	
$\|V\|$	p	$\|E\|$	Time	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

Table 7 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 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 8, 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 8: The effect of weight distribution on Algorithm B and the Babel Algorithm

				B	Babel	
$\|V\|$	p	$\|E\|$	Interval	Time	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 forebears, we now wish to consder their usefulness with respect to all existing algorithms. To that end, we will use Algorithm AB, the hybrid described in Section 6.1. 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 result in a faster algorithm overall (but note that, for some graphs, even its branch-and-bound tree is smaller than those of both Algorithms A and B). Tables 9 and 10 shows how Algorithm AB performs on the graphs we have encountered so far.

Table 9: Algorithms B and AB on DIMACS graphs

				Babel/B		AB	
Graph	$\|V\|$	$\|E\|$	$\alpha(G)$	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

Table 10: 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

We will compare Algorithm AB to Cliquer (see `users.tkk.fi/~pat/-cliquer.html`), a freely available implementation of both the algorithm in Östergård (2002) and an extension of it to the maximum weight independent set 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. 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 as often as possible. When a random selection is possible, randomly choose a vertex 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 11 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 11: 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

Table 12 gives results comparing these two algorithms on graphs whose vertices have one of two degrees. For each pair a, b under the heading "Degree", 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.

Table 12: 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

8. Conclusions

We recognized that what we call the Chordal Method used in Balas and Yu (1986) to generate clique covers serves a primarily heuristic function in their overall algorithm. Its inferiority to what we call the Greedy Method of Balas and Yu (1986) was demonstrated by our experiments in Section 4. The branching scheme in Balas and Yu (1986) 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 (Babel, 1994). 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 root node of the branch-and-bound tree with the speed of Method A at the child nodes. Algorithm AB outperforms the state-of-the-art algorithm Cliquer on some graphs.

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. They 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 the sensitivity of Algorithm AB's performance relative to that of Cliquer depending on graph structure, it seems profitable to test these algorithms on other varieties of graphs.

Acknowledgments

This research was partially supported by NSF Grant DMI-0217265.

References

Babel, L. 1994. A fast algorithm for the maximum weight clique problem. *Computing* **52** 31–38.

Balas, E., J. Xue. 1991. Minimum weighted coloring of triangulated graphs, with application to maximum weight vertex packing and clique finding in arbitrary graphs. *SIAM J. Comput.* **20** 209–221.

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

Balas, E., C. S. Yu. 1986. Finding a maximum clique in an arbitrary graph. *SIAM J. Comput.* **15** 1054–1068.

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

Bomze, I. M., M. Budinich, P. M. Pardalos, M. Pelillo. 1999. The maximum clique problem. D. Du, P. M. Pardalos, eds., *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers, Boston, 1–74.

Bourjolly, J.-M., G. Laporte, H. Mercure. 1997. A combinatorial column generation algorithm for the maximum stable set problem. *Oper. Res. Lett.* **20** 21–29.

Brélaz, D. 1979. New methods to color the vertices of a graph. *Comm. of the ACM* **22** 251–256.

Bron, C., J. Kerbosch. 1973. Finding all cliques in an undirected graph. *Comm. of the ACM* **16** 575–577.

Carraghan, R., P. M. Pardalos. 1990. An exact algorithm for the maximum clique problem. *Oper. Res. Lett.* **9** 375–382.

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Company, New York.

Johnson, D. S., M. A. Trick, eds. 1996. *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series in Discrete Math. and Theoret. Comput. Sci.*, vol. 26. Amer. Math. Soc.

Mannino, C., A. Sassano. 1996. Edge projection and the maximum cardinality stable set problem. Johnson and Trick (1996), 205–219.

Mehrotra, A., M. A. Trick. 1996. A column generation approach for graph coloring. *INFORMS J. Comput.* **8** 344–354.

Östergård, P. R. J. 2002. A fast algorithm for the maximum clique problem. *Discrete Appl. Math.* **120** 197–207.

Rose, D. J., R. E. Tarjan, G. S. Lueker. 1976. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.* **5** 266–283.

Rossi, F., S. Smriglio. 2001. A branch-and-cut algorithm for the maximum cardinality stable set problem. *Oper. Res. Lett.* **28** 63–74.

Sewell, E. C. 1998. A branch and bound algorithm for the stability number of a sparse graph. *INFORMS J. Comput.* **10** 438–447.

West, D. B. 2001. *Introduction to Graph Theory*. 2nd ed. Prentice Hall, Englewood Cliffs, New Jersey.

Wood, D. R. 1997. An algorithm for finding a maximum clique in a graph. *Oper. Res. Lett.* **21** 211–217.