A Branch-and-Price Approach for the Maximum Weight Independent Set Problem

Deepak Warrier (deepakwarrier@hotmail.com)
Wilbert E. Wilhelm (wilhelm@tamu.edu)[1]
Jeffrey S. Warren (j-warren@tamu.edu)
Illya V. Hicks (ivhicks@tamu.edu)

Department of Industrial Engineering
Texas A&M University
TAMUS 3131
College Station, TX 77843-3131

March 9, 2005 Revised May 31, 2005 August 22, 2005

Abstract

The maximum weight independent set problem (MWISP) is one of the most well-known and well-studied problems in combinatorial optimization. This paper presents a novel approach to solve MWISP exactly by decomposing the original graph into vertex-induced sub-graphs. The approach solves MWISP for the original graph by solving MWISP on the sub-graphs in order to generate columns for a branch-and-price framework. The authors investigate different implementation techniques that can be associated with the approach and offer computational results to identify the strengths and weaknesses of each implementation technique.

Key Words

Branch and price, maximum weight independent set problem, Dantzig-Wolfe Decomposition, vertex partitioning, inheritly decomposable graphs

[1] Communicating author

1 Introduction

A set of a graph's vertices is an *independent set* if no two vertices in the set are adjacent (i.e., connected by an edge). Given a weighting of vertices, the *maximum weight independent set problem* (MWISP), which is NP-hard [23], is to prescribe an independent set of the graph that has maximum weight. MWISP has many important applications, including combinatorial auctions [44], graph coloring [34], coding theory [32], geometric tiling [18], fault diagnosis [11], pattern recognition [26], molecular biology [22, 24, 35], and scheduling [27]. Additional applications arise in more comprehensive problems that involve MWISP with side constraints.

This paper investigates a novel approach that solves MWISP exactly. Our approach partitions the vertex set of a graph to obtain smaller vertex-induced sub-graphs on which MWISP is less challenging, on average, to solve. We use a branch-and-price (B&P) framework to solve MWISP on the original graph G using these specially constructed sub-problems to generate columns. MWISP is a member of the class of graph problems that we call *inheritly decomposable* because instances can be decomposed with sub-problems of the same type as the original problem (i.e., they inherit the nature of the original problem).

Our goal is to investigate methods to distribute and manage the challenges posed by complexity. In a sense, a traditional application of B&P deals with complexity by decomposing a block diagonal matrix structure, distributing the challenges of complexity among sub-problems and the master problem. However, the analyst typically has little control over the sizes and structures of the sub-problems. In contrast, our approach allows the analyst to distribute complexity by prescribing the sizes and structures of sub-problems and manage it, for example, by devising algorithms to solve resulting sub-problems, facilitating solution of the master problem (e.g., reducing degeneracy and improving the co-ordination of sub-problem solutions), implementing

special-purpose branching rules, and tightening the master problem to facilitate solution. Specific objectives of this research are to present a rationale for using price-directed decomposition to solve MWISP, to investigate effective implementation techniques, and to conduct computational tests to identify strengths and weaknesses of the approach.

This paper deals only with finite, simple graphs that are undirected. In graph $G=(V,E)$, which comprises vertex set V and edge set E, an edge in E joining vertices $u,v\in V$ is denoted uv. The *neighbors* of vertex $v\in V$ are $N(v) = \{u\in V : uv\in E\}$ and the non-neighbors of v are $\bar{N}(v)=V\setminus\left(N(v)\cup\{v\}\right)$. We extend this notation to a set $W\subseteq V$ by defining $N(W)=\left(\cup_{v\in W}N(v)\right)\setminus W$ and $\bar{N}(W)=V\setminus\left(N(W)\cup W\right)$. For any $W\subseteq V$, we denote the sub-graph induced by W as $G[W]$; and for any $F\subseteq E$, we denote the sub-graph induced by F as $G[F]$. A non-negative weight w_v is associated with vertex $v\in V$. We use $S\subseteq V$ to denote an *independent set* of G and sub-graph K to denote a *clique* of G (i.e., a complete sub-graph of G). For an introduction to graph theory, we refer the reader to [45].

Bron and Kerbosch [15] initiated exact approaches to the *maximum independent set problem*, a special case of MWISP with $w_v=1$ for all $v\in V$, by proposing an explicit enumeration procedure. Subsequent improvements using branch-and-bound (B&B) [4, 17] stimulated development of optimization methods for MWISP [1, 2, 16, 19, 28, 33, 38, 39, 40, 41, 47]. Balas and Xue [3] devised a notable algorithm that extended their previous work [2], using a fast heuristic for the weighted-fractional coloring problem to obtain good upper bounds for B&B. Other effective heuristics are also available [16]. Nemhauser and Sigismondi [36] and Verweij [42, 43] have reported applications of strong cutting plane methods. Mehrotra and Trick [34] recently proposed a column generation approach to the *minimum coloring problem* that employs a set-covering master

problem and a single sub-problem, which generates columns, each of which is a solution to MWISP with objective function coefficients determined by the values of dual variables associated with the master problem.

The body of this paper is organized in six sections. Section 2 presents the decomposition of MWISP that our B&P approach uses to distribute the challenges of complexity. Sections 3, 4, and 5 describe methods for managing the challenges of complexity. Section 3 discusses two methods for partitioning a graph to create the sub-problems required by our approach. Section 4 relates two methods for managing the size of the restricted master problem (RMP) (so called because it deals with a restricted set of all possible columns), and Section 5 describes two special-purpose branching rules. Section 6 summarizes our computational tests and Section 7 presents conclusions and recommendations for future research.

2 B&P Formulation

MWISP can be formulated as a linear integer program:

$$Z^*_{MWISP} = \text{Max}\left\{\sum_{v\in V} w_v x_v : \boldsymbol{x} \in Q\right\},$$

in which edge inequalities define Q:

$$Q = \left\{\boldsymbol{x} \in B^{|V|} : x_u + x_v \le 1, \forall uv \in E\right\}, \qquad (2)$$

where B^n denotes the set of binary n vectors ($n = |V|$ in this case) and

$x_v = 1$ if vertex *v* is included in the independent set and $x_v = 0$ otherwise.

Our approach begins by partitioning the vertex set of graph $G = (V, E)$ into P parts, $V_1, \ldots, V_P$. For every $p \in \{1, \ldots, P\}$, we define sub-graph $G_p = G[V_p]$ with edge set $E_p = E(G_p)$. Edges of G

whose ends lie in different sets of the partition constitute set $\hat{E} = E \setminus \bigcup_{p=1}^{P} E_p$, which induces the sub-graph $G(\hat{E})$ with vertex set that we denote by $\hat{V}$.

With this partitioning, MWISP can be expressed as

$$Z^*_{MWISP} = \text{Max}\left\{ \sum_{p=1}^{P} \sum_{v \in V_p} w_v x_v : x_u + x_v \le 1, \forall uv \in \hat{E}, \mathbf{x}^p \in Q_p, \forall p \in \{1,...,P\} \right\}, \qquad (3)$$

where $Q_p = \left\{ \boldsymbol{x} \in B^{|V_p|} : x_u + x_v \le 1, \forall uv \in E_p \right\}$. Formulation (3) has a block-diagonal structure that is amenable to B&P:

$$Z^*_{\text{MWISP}} = \text{Max} \sum_{v \in V} w_v x_v = \text{Max} \sum_{p=1}^{P} \mathbf{w}^p \mathbf{x}^p$$

subject to

$$\begin{bmatrix} A_1 & A_2 & \cdots & A_P \\ D_1 & 0 & \cdots & 0 \\ 0 & D_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D_P \end{bmatrix} \begin{pmatrix} \boldsymbol{x}^1 \\ \boldsymbol{x}^2 \\ \vdots \\ \boldsymbol{x}^P \end{pmatrix} \le \mathbf{1} \text{ and } \mathbf{x}^p \in B^{|V_p|}, \ \forall p \in \{1,\ldots,P\},$$

where A_p: matrix of coefficients in inequalities associated with edges $uv \in \hat{E}$

D_p: matrix of coefficients in inequalities associated with edges $uv \in E_p$

$\mathbf{x}^p$: $|V_p|$-vector of decision variables associated with vertices $v \in V_p$

$\mathbf{w}^p$: $|V_p|$-vector of weights associated with vertices $v \in V_p$.

Applying Dantzig-Wolfe Decomposition [5] to the linear relaxation of (3), we obtain RMP, a restricted version of the master problem that includes a subset of the columns associated with the extreme points of sub-problem polytopes:

$$Z^*_{\text{LP}} = \text{Max} \sum_{p=1}^{P} \sum_{j \in J_p} \lambda_{jp} (\mathbf{w}^p \mathbf{x}^{jp})$$

subject to

$$\sum_{p=1}^{P} \sum_{j \in J_p} \lambda_{jp} (A_p \mathbf{x}^{jp}) \quad \leq 1 \tag{4}$$

$$\sum_{j \in J_p} \lambda_{jp} \quad = 1 \qquad \forall \ \ p \in \{1, \ldots, P\} \tag{5}$$

$$\lambda_{jp} \quad \geq 0 \qquad \forall \ \ p \in \{1, \ldots, P\}, \ \ j \in J_p,$$

where J_p: set of integer extreme points of Q_p associated with generated columns in RMP

$\mathbf{x}^{jp}$: $\left|V_p\right|$-vector that defines extreme point $j \in J_p$

λ_{jp}: RMP decision variable that corresponds to extreme point $j \in J_p$.

MWISP sub-problem $p \in \{1, \ldots, P\}$ has the form

$$Z^*_p = \text{Max} \left\{ \left(\boldsymbol{w}^p - A_p^T \boldsymbol{\alpha} \right) \boldsymbol{x}^p : \boldsymbol{x}^p \in Q_p \right\},$$

where $\boldsymbol{\alpha}$ is the $|\hat{E}|$-vector of dual variables associated with the rows of constraints (4). In this model, $\mathbf{x}^{jp}$ is an improving column if $Z^*_p - \beta_p > 0$, where β_p is the dual variable associated with the p^{th} convexity constraint (5). The role of RMP is to co-ordinate sub-problem solutions (through the values of dual variables) to prescribe a solution to MWISP on graph *G*. Depending on the sub-graph structure, the sub-problem may also be NP-hard; however, sub-graphs are smaller than the original graph G and we expect that they will be less challenging, on average, for optimizing algorithms to solve.

For an introduction to column generation, we refer the reader to [46]. We manage our column pool and sub-problem optimization in standard ways (again, see [46]), for example,

entering, at each iteration, all improving columns into the column pool that CPLEX manages.

3 Partitioning *V*(*G*)

We use two alternative methods to partition *V*(*G*). The first method (p1) employs the procedure of Balas and Yu [4] to find a maximal chordal induced sub-graph G_1 in *G*. We then find a maximal chordal induced sub-graph G_2 in $G\left[V\left(G\right)\backslash V\left(G_1\right)\right]$ and repeat the process until every vertex of *G* has been included in a chordal induced sub-graph. The advantage of (p1) is that MWISP can be solved on each chordal sub-graph in polynomial time, for example, using Frank's solver [21]. A prime disadvantage is that it results in a large number of small sub-problems so that $\left|\hat{E}\right|$ is large and, consequently, RMP is also large and requires more computational effort.

The second method (p2) applies METIS [29-31], a well-known clustering heuristic, to partition *V*(*G*) into a predetermined number P of sets. METIS assigns vertices to sets so that the cardinalities of the sets differ as little as possible (i.e., to assign *n* vertices to *k* sets, it creates $(n \bmod k)$ sets of $\lceil \frac{n}{k} \rceil$ vertices and $k-(n \bmod k)$ sets of $\lfloor \frac{n}{k} \rfloor$ vertices). Subject to that constraint, METIS attempts to minimize the number of edges that have ends in different sets. As many as $P=\left|V\right|$ sets can, in principle, be specified for (p2), but we determine P by setting an *a priori* bound based on the size of a sub-problem that can typically be solved in an acceptable amount of time. The advantage of (p2) is that it gives the analyst more control over the number of sets *P* and the density of $\hat{G}$; its primary disadvantage is that the induced sub-graphs cannot be expected to have a special structure so that, typically, sub-problems cannot be solved in polynomial time. We adapted the Carraghan-Pardalos algorithm [17], which finds a maximum clique in a graph, to solve these sub-problems.

The resulting Carraghan-Pardalos Adapted Algorithm (CPAA) incorporates obvious adaptations to enumerate all maximal independent sets (which suffices when all vertex weights are non-negative) either explicitly or implicitly by pruning one of its ancestors in a search tree whose nodes represent all independent sets of G. Next, we outline our adaptation, CPAA.

First, index the vertices of G as $\{v_1,\ldots,v_n\}$. The root node of the search tree represents the empty set, but each of the other nodes in the tree represents an independent set containing all vertices that comprise its parent's independent set plus one additional vertex whose index (according to our numbering) is greater than those in its parent's independent set. All children of a node in the search tree that represent independent sets must be considered to ensure that all independent sets are enumerated. Since each leaf of the tree represents a maximal independent set, enumerating leaves of the tree corresponds to enumerating maximum weight independent sets (assuming, as we do, that all vertex weights are non-negative). Because the number of maximal independent sets can be prohibitively large, searching major parts of the tree implicitly is most desirable.

Begin a search by assigning the heaviest known independent set to be empty (i.e., $\hat{S} = \varnothing$) and activating the root node in the search tree. An active non-leaf node of the tree represents an independent set S to which we assign a weight that is the sum of the weights of vertices in $S \cup \overline{N}(S)$. If this exceeds the weight of $\hat{S}$, construct the children of that node and make one of them active. Otherwise, eliminate the sub-tree rooted at the current node and set as active its nearest unexplored relative (sibling, parent's sibling, grandparent's sibling, etc.), if any exists. If an active leaf of the tree represents an independent set S that is heavier than $\hat{S}$, update $\hat{S} \leftarrow S$, eliminate the sub-tree rooted at the current node, and designate as active the nearest unsearched relative of the current node, if any exists. After enumerating the search tree in this manner until no additional nodes can be made active, $\hat{S}$ prescribes a maximum weight independent set of G. In

addition to using CPAA to solve sub-problems, we use it in our computational tests to solve each MWISP instance as a basis of comparison with our approach.

4 RMP

Method (p1) results in a large number of edges whose ends lie in different sets of the partition (i.e., $\left|\hat{E}\right|$ is large) so that RMP has a large number of rows. Even though method (p2) employs METIS, which seeks to minimize $\left|\hat{E}\right|$, it may also result in a large RMP, especially in application to a dense graph. This leads to a second issue because it can be shown that edge inequalities (4) relegated to RMP form a polytope that is degenerate if any vertex is included in more than one inequality. If a vertex appears in *m* edge inequalities, it increases the order of degeneracy of RMP by $m-1$. Method (p1) typically relegates many edge inequalities to RMP and causes most vertices to appear in several of these inequalities, so it leads to large, highly degenerate RMPs. If *K* is a clique in *G* that induces *m* edges of $\hat{E}$, replacing the *m* corresponding constraints in RMP by a single inequality of the form $\sum_{v\in V(K)} x_v \le 1$ would decrease the order of degeneracy of RMP by $2m-\left|V(K)\right|$, assuming $V(K)\subseteq\hat{V}$ (we can substitute $V(K)\cap\hat{V}$ for $V(K)$ without reducing *m*). Note, however, that for a second clique K' that induces m' edges in $G\left[\hat{E}\right]\backslash V(K)$ and m'' edges in $\hat{E}$, substituting the clique inequality would reduce RMP degeneracy by $2m'-\left|V(K')\right| \le 2m''-\left|V(K')\right|$. All degeneracy could be eliminated by such substitutions only if every component of $G\left[\hat{E}\right]$ is a clique, an unlikely circumstance.

We dealt with RMP using two alternative methods. The first method (m1) simply uses the constraints associated with edges in $\hat{E}$; it offers the advantage of simplicity but incurs the

computational disadvantages caused by the resulting size of RMP and its attendant degeneracy.

The second method (m2) solves a set-covering problem using a greedy heuristic [37] to select a set of clique and edge inequalities that covers all edges in $\hat{E}$. To identify clique inequalities we employ a best-in greedy heuristic, which the next paragraph describes, to construct a clique $\hat{K}$ that is maximal in $G[\hat{E}]$ and then employ the same heuristic to find a maximal clique K of $G[\hat{V}]$ such that $V(\hat{K}) \subseteq V(K)$. This method offers the advantages of producing a smaller polytope than does method (m1), cutting off many fractional solutions that are feasible for the (m1) polytope; of reducing the size of RMP in comparison with method (m1); and of incorporating clique inequalities that tighten RMP, improving the bounds it prescribes. However, it may cover some edges of $\hat{E}$ with more than one inequality, so that it may not eliminate degeneracy altogether.

The best-in greedy heuristic, which we employ to find independent sets, uses the function η_G to assign a value to each vertex $v \in V$ equal to the ratio of its weight to the weight of its neighbors: specifically, it starts by setting $S = \varnothing$ and assigning $\eta_G(v) = w(v) / w(N_G(v))$, where $w(N_G(v)) = \sum_{u \in N_G(v)} w(u)$. At each iteration, it selects a vertex $\hat{v} = \arg\max_{v \in V} \{\eta_G(v)\}$ and includes $\hat{v}$ in the independent set, $S \leftarrow S \cup \{\hat{v}\}$; removes the neighbors $N(\hat{v})$ of $\hat{v}$ from consideration; and updates the values $\eta_G(v)$ for $v \in \bar{N}(\hat{v})$. It implements this process at each iteration until the set of selected vertices S form a *maximal* independent set. We use a complementary procedure to find heavy cliques (see Section 5).

5 Branching

To obtain a lower bound for MWISP at each node in the search tree, we select the k vertices

with the highest η_G values and apply our best-in heuristic to each of k graphs $G\left[\bar{N}(v)\right]$, where v is one of these k vertices. For each independent set S prescribed by the heuristic, $S \cup \{v\}$ is a maximal independent set in G; we use the weight of the heaviest of these k sets as an initial lower bound for MWISP. We then solve RMP to optimality to obtain an upper bound for MWISP at that node.

We use two different branching rules when the optimal solution of RMP is fractional. The first rule (b1) branches on the most fractional variable $x_v = \sum_{j \in J_p} x_v^{jp} \lambda_{jp}$ where $v \in V_p$. Two new (child) B&B nodes result: one in which $x_v = 0$, and one in which $x_v = 1$. This traditional branching rule offers the advantage that it has been used successfully in many situations.

The second rule (b2) branches on the vertices of a clique. We identify a clique for branching by weighting each vertex according to the fractional part of its associated variable (i.e., the weight of vertex v is given by $\left(0.5 - \left|x_v - 0.5\right|\right)$) and then greedily constructing a clique *K* of large weight that includes no vertices whose variables are fixed in the current B&B node. To branch on the clique (see successful, analogous methods in [6, 7, 8, 9, 10, 20]), we create $\left|V(K)\right| + 1$ children, fixing $x_v = 1$ for one $v \in K$ at each child node $1, ..., \left|V(K)\right|$ and all variables to zero at child node $\left|V(K)\right| + 1$. Our sub-problem solvers enforce the fixed values of decision variables. Preliminary tests indicate that this method improves on traditional SOS branching [37] applied to the same set. Note that *K* need not be maximal and need not correspond to a cut for the current fractional solution.

6 Computational Benchmarks

This section describes the computational environment, sets of instances, and measures of performance we use in our tests. A table displays the results of each test and we discuss them to put them into perspective.

We conduct all tests using a Dell Optiplex GX240 with a 3-GHz Pentium IV processor and 512 MB of memory. We use CPLEX 7.1 to solve the linear programs.

We employ two sets of test instances. The first set comprises instances from the Second DIMACS Implementation Challenge, which are unweighted (we actually use the complements of the listed graphs, and this is reflected in the values of $|E|$). These DIMACS instances represent several families of graphs, each with a special structure induced by a particular application.

Second, we generate random π graphs to study our methods in application to instances that have no special structure. After specifying the number of vertices $|V(G)|$ and the parameter π, we generate each π graph *G* by including each possible edge (denoted *uv* and *vu*) with probability π $(0 \le \pi \le 1)$. We then draw the weight of each vertex from a discrete uniform distribution on the interval $[1, M]$, with *M* = 1, 20, 40, 60, or 100. The *M* = 1 case is, of course, the (unweighted) independent set problem. For each value of π, we generate 25 independent instances (each using a unique random number seed), forming five subsets, each with a different value of *M* and each comprising five instances. We note that very sparse graphs (i.e., densities of 5% or less) are not likely to be connected; we use them in our tests to evaluate the robustness of our B&P approach in comparison with CPAA. Very sparse graphs are encountered in practice (e.g., Hamming8-2 in Table 4) so robustness is a desirable characteristic of a solution approach.

We describe the graph involved in each test, giving the number of vertices $|V|$, the number of edges $|E|$, and the % Density $\Delta = \lceil (200)|E| \rceil / \lceil |V|(|V|-1) \rceil$. We relate *P*, the number of sets

we specified in the partition, and the resulting $|\hat{E}|$ and RMP Rows, the number of rows in RMP. To evaluate model tightness, we give Z_{LP}, Z_H, and Z_{IP} (or appropriate ratios), the optimal value of RMP at the root node of the B&B tree, the value that the initial heuristic returns, and the optimal value of the integer program, respectively. Finally, we use several measures of performance to evaluate our B&P approach: B&B Nodes, the number of B&B nodes required to find the optimal solution; MP Sols., the total number times RMP is solved; and B&P Time, the CPU run time for our B&P approach to prescribe an optimal solution.

We started with a battery of preliminary tests to compare the performances of alternative methods (p1) and (p2), (m1) and (m2), and (b1) and (b2). Of the 8 possible combinations of these 3 pairs of alternative methods, we report results on the same set of 13 DIMACS instances for the 3 most promising combinations in Tables 1-3.

Table 1 compares performances of (m1) and (m2) using methods (p2) and (b2). The first six columns in Table 1 define the instance: graph designation, $|V|$, $|E|$, Δ, P, and $|\hat{E}|$. The last five columns give the method (i.e., (m1) or (m2)) and the results for each, including RMP Rows, B&B Nodes, MP Sols., and B&P Time. Results show that (m2) solves 7 of these 13 instances faster than (m1) (including 3 of the 4 most challenging instances), essentially ties on 5 of the instances, and is substantially slower on only one instance (brock200-3). Because (m2) must incur a "set up" time to identify cliques, it is at somewhat of a disadvantage, especially on relatively sparse and relatively dense instances, which tend to require longer times for this set up. (m2) requires more B&B nodes than (m1) in only 2 instances (brock200-3 and p_hat300-1), indicating that (m2) typically leads to a tighter model. (m2) yields a smaller RMP (see RMP Rows) in 11 of the 13 instances and ties in the remaining two. This speeds solution time in two ways: a smaller RMP basis requires less computational effort and, significantly, less degeneracy is involved. The advantage provided by a

smaller RMP may be substantial; for example, the p_hat300-1 instance (m2) requires considerably more B&B Nodes and MP Solutions, but it requires less run time because it yields a smaller RMP. Method (m1) outperforms (m2) substantially in only one instance (brock200-3). Based on this comparison, we select (m2) as a default to use on other tests.

Table 2 compares performances of (p1) and (p2) using methods (m2) and (b2). Results show that (p2) solves 12 of the 13 instances faster than (p1) and essentially ties (p1) on the 13^{th} instance (hamming8-2). (p1) suffers from a property that is difficult to overcome. In particular, MWISP on a sub-graph that results from (p1) may be modeled using edge inequalities as shown in (2). The polytope of the linear relaxation of (2) contains the convex hull of feasible integer solutions. However, chordal graphs are perfect (West [45]) so that associated clique inequalities can be used to model MWISP; the polytope of the linear relaxation in this case is the same as the convex hull of feasible integer solutions. Thus, (p1) defines sub-problems that have the Integrality Property (Wilhelm [46]); it is well known that such formulations may offer the advantage that sub-problems can be solved in polynomial time but they typically suffer the serious disadvantage that they do not prescribe tight bounds and, therefore, more nodes must be explored in the B&B tree, requiring longer solution times. We conclude that (p2) outperforms (p1) because it yields sub-problems that do not, in general, exhibit the Integrality Property, resulting in smaller, tighter RMPs. B&B Nodes shows that (p2) enjoys a considerable advantage from the tightness of RMPs. Method (p1) typically results in more sub-problems so that $\left|\hat{E}\right|$ and, consequently, RMP rows are larger, requiring more run time.

Table 3 compares the performances of (b1) and (b2) using methods (p2) and (m2). Results show that method (b2) is faster than (b1) on 10 of the 13 instances; it is significantly faster on each of the 4 most challenging instances. Method (b2) essentially ties (b1) on the remaining three

instances, which are small graphs for which the overhead involved in finding cliques puts (b2) at a disadvantage. Although (b2) creates more children at each node in the branch-and-bound tree than (b1), (b2) requires much smaller search-trees than (b1), on average (see B&B Nodes). We conclude that (b2) is superior because cliques promote more discerning branching, improving performance, even though (b2) requires time to identify appropriate cliques.

These tests show that a number of complex interactions influence the performance of our B&P approach. Based on preliminary tests, which are only partially described above, we conclude that the (m2)-(p2)-(b2) combination is the most reasonable and intuitively appealing.

Next, we evaluate the (m2)-(p2)-(b2) combination more extensively. Table 4 compares the performance of our B&P approach to that of CPAA on the set of 13 DIMACS instances. We test three different values of P on each instance to evaluate the sensitivity of our B&P approach to that parameter. We selected CPAA as a basis of comparison because the Carraghan-Pardalos algorithm is effective, easily implemented, and has been used for benchmarking previously (e.g., see Balas and Xue [2]). Columns 1-4 in Table 4 describe each DIMACS instance: graph designation, $|V|$, $|E|$, and Δ. Columns 5-7 give the value of P we selected and the resulting $|\hat{E}|$ and RMP Rows, respectively. Columns 8-10 list Z_{LP}, Z_H, and Z_{IP}, respectively. Finally, Columns 11-14 give the performance measures we use: B&B nodes; MP Sols.; B&P Time; and CPAA Time, the CPU run time for CPAA to solve the instance.

Results show that the performance of our B&P approach is sensitive to *P*, the number of sets specified in the partition; that it is able to solve the Hamming and Johnson graphs at the root node; and that it is more effective than CPAA on graphs with densities less than 40%. Method (m2) reduces the RMP size substantially for many of these graphs (compare $|\hat{E}|$ and RMP Rows). Our run times to solve these DIMACS instances are quite reasonable, even for graphs with a large

number of vertices.

In preliminary tests, we selected P using the rule of thumb that each sub-graph should comprise 30–50 vertices, a size that CPAA can typically solve effectively. We found that CPAA cannot deal effectively with large, sparse sub-graphs so we increased the value of P for sparse graphs to make sub-graphs smaller. However, as P increases, $\left|\hat{E}\right|$ also increases because more edges connect vertices in two different partitions. For larger values of P, sub-graphs are smaller so that sub-problems contain fewer edge inequalities, RMP provides weaker bounds, and the gap (i.e., the difference between the optimal solutions of the integer program and its linear relaxation) is larger. We conclude that smaller $\left|\hat{E}\right|$ is good in two ways: it indicates that RMP is smaller and easier to solve and sub-problems contain more edge inequalities so that RMP provides tighter bounds. Based on preliminary tests, we select $P = 4$ as a reasonable "default" value for all instances, although some can be solved much faster by selecting more appropriate P values.

Table 5 describes the random π graphs that we generate for testing; all graphs use $|V| = 100$ and $P = 4$; our analysis and conclusions relate to this set of graphs and do not attempt to extrapolate to other cases. Column 1 gives the value of π and other columns describe the generated set of instances. Columns 2-4, 5-7, and 8-10 give minimum, maximum and average values (over five instances) for the generated value of $|E|$ and the $|\hat{E}|$ and RMP Rows that result from method (m2) for the specified $|V|$, P, and π, respectively.

Method (m2) reduces the size of the master problem if cliques that include more than two vertices that are incident to edges in $\hat{E}$ can be found. If our procedure identifies a clique that includes only two vertices connected by an edge in $\hat{E}$, it does not reduce the number of rows in the master problem. Only a few cliques identified for the sparse graphs in Table 5 include more than

two vertices connected by an edge in $\hat{E}$, so that RMP Rows is not much less than $\left|\hat{E}\right|$ for these instances (i.e., π = 0.01, 0.05, and 0.10). In any event, clique inequalities tighten the master problem. Thus, clique inequalities can improve run time in two ways: by reducing the size of the master problem and by tightening it. We employ our best-in heuristic to identify cliques but a more sophisticated procedure may be able to find "better" cliques for (m2) to use.

Table 6 reports test results. Column 1 gives π, providing a cross-reference to corresponding measures in Table 5, and column 2 relates M. We solve each generated graph for the unweighted case and then solve 4 additional instances, each using a different distribution for assigning weights to vertices. Columns 3-6 give overall measures, including Z^*_{LP} / Z^*_{IP}, Z_H / Z^*_{IP}, B&B nodes, and RMP iterations. Columns 7-9 give the minimum, maximum and average run times for our B&P approach to solve each set of five random graphs. Columns 10-12 give corresponding run times for CPAA to solve each set of five random graphs.

As π increases, the upper bounds from the linear relaxations (Z^*_{LP} / Z^*_{IP} values in column 3) as well as the lower bounds from the heuristic (Z_H / Z^*_{IP} values in column 4) degrade. The tightness of Z^*_{LP}, the optimal solution at the root node, reduces as π increases because sub-graphs contain fewer edges (they are assigned to $\hat{E}$) so that generated columns reflect fewer (edge inequality) constraints. As expected, weaker bounds make the denser problems more challenging (note B&B Nodes in column 5, RMP iterations in column 6 and run times in columns 7-9).

Results in columns 5 through 12 show that, for a given π, the set of unweighted instances is consistently more challenging than the set of related, weighted instances. Weighted instances (i.e., with M = 20, 40, 60, 100) have comparable run times for most values of π (exceptions are for $\pi = 0.05$ and for $M = 100$ with $\pi = 0.10$ and $\pi = 0.15$).

Most of the random graphs we generate using $\pi = 0.01$ are disconnected; in these instances METIS is able to partition the vertices so that $|\hat{E}|$ = RMP Rows = 0 (see Table 5). Our B&P approach solves MWISP at the root node of the B&B tree in each instance with $\pi = 0.01$ (see Table 6).

For each value of M there is a trend for average run time to increase monotonically up to a certain value of π, then to decrease monotonically as π continues to increase. For example, this value of π is 0.20 for the $M = 1$ (i.e., unweighted) case. CPAA failed to solve all instances with $0.01 \le \pi \le 0.10$ because they exceeded memory capacity (512 MB). Our B&P gives better run times for instances with $0.01 \le \pi \le 0.20$, but CPAA requires less run time on denser instances with $0.30 \le \pi \le 0.50$. Even in these denser cases, however, our B&P approach does not require excessive run times.

Complex interactions cause the run time of our B&P approach to increase, then decrease, as π increases. When π is small, $|\hat{E}|$ is small so that RMP has few rows and a low order of degeneracy. Even though CPAA is most effective in application to dense (sub)graphs, sub-problem solutions need little co-ordination so that B&P requires little run time to solve sparse problems. $|\hat{E}|$ increases with π, increasing the number of rows in RMP, the order of degeneracy of RMP, and run times. As π increases further, $|\hat{E}|$ and the number of rows in RMP and its order of degeneracy all continue to increase but CPAA can solve the resulting sub-problems, which are quite dense, effectively. In addition, sub-problems provide more information about the global solution because they incorporate more edges; as a consequence, RMP requires fewer iterations to prescribe an optimal solution.

7 Conclusions

This paper offers a new approach for solving MWISP by utilizing a price-directed decomposition approach in conjunction with a branch-and-price framework to divide the complexity of the problem between the master problem and sub-problems. This paper also offers a computational evaluation of the approach and accompanying implementation techniques to offer guidance for utilizing the approach.

We evaluate two methods for partitioning the graph, (p1) and (p2); two methods for dealing with the master problem, (m1) and (m2); and two methods for branching, (b1) and (b2). Preliminary computational tests indicate that the (p2), (m2), and (b2) combination is generally more effective than alternatives.

Tests using a set of graphs taken from the Second DIMACS Implementation Challenge and another set of randomly generated π graphs show that our B&P approach is more effective in application to sparse graphs, which result in small RMPs with low orders of degeneracy. These tests on random π graphs also show that the unweighted maximum independent set problem is consistently more computationally challenging than its MWISP counterpart. Run time is sensitive to P, the number of sets in the partition. Parameters π and P affect the magnitude of $|\hat{E}|$ and, thus, the size of the master problem, which increases with $|\hat{E}|$, tending to require longer run times. Importantly, our B&P approach is effective in application to graphs of low densities, the category of instances that are most challenging for earlier approaches. In particular, our B&P approach performs well on very sparse graphs, demonstrating its robustness in comparison with CPAA. These conclusions relate to the set of graphs we tested with $|V| = 100$ vertices and $P = 4$ partitions; we do not attempt to extrapolate to other cases.

Clearly, the challenges of complexity cannot be eliminated. This paper shows, however, that they can be distributed and managed to improve performance. By investigating two methods to

partition a graph, this paper explores alternatives for distributing the challenges of complexity by prescribing the sizes and structures of sub-problems. By studying two methods to formulate the master problem and two branching rules, this paper assesses means of managing these challenges. Specific techniques used to this end include controlling the size and tightness of the master problem, devising algorithms to solve resulting sub-problems, facilitating solution of the master problem (e.g., reducing degeneracy and improving the co-ordination of sub-problem solutions), implementing special-purpose branching rules, and tightening the master problem to facilitate solution. The tests described in this paper demonstrate that a complex set of trade offs must be accommodated in managing these challenges, including the sizes and structures of sub-problems; the size, tightness, and order of degeneracy posed by the master problem; the size and density of the graph induced by edge inequalities that are relegated to the master problem; and the capabilities of the algorithm(s) employed to solve sub-problems and the overall B&B search.

Future research could contribute, for example, by devising techniques to determine *a priori* an optimal number of partitions (i.e., value of P) for a particular instance. Other fertile opportunities to advance this line of research include devising more effective methods for partitioning, for dealing with large degenerate master problems, and for incorporating cutting planes to tighten the master problem in a branch-and-price-and-cut approach. In addition, this study indicates an attractive potential for the successful application of B&P to other inheritly decomposable graph problems. Our research continues along these lines.

Acknowledgement

This material is based upon work supported by the National Science Foundation on Grant number DMI-0217265 and by the Texas Advanced Technology Program under Grant Number 000512-0248-2001. The authors acknowledge the able assistance of Chris Gillard and note their

indebtedness to two anonymous referees and an Associate Editor, who offered comments that allowed us to strengthen an earlier version of this paper.

8 References

[1] E. Balas, S. Ceria, and G. Cornuejols, Mixed 0-1 programming by lift and project in a branch and cut framework, Management Science 42 (1996), 1229-1246.

[2] E. Balas and J. Xue, Minimum weighted coloring of triangulated graphs with applications to maximum weight vertex packing and clique finding in arbitrary graphs, SIAM Journal on Computing 20 (1991), 209-221, (Addendum, SIAM Journal on Computing 21 (1992), 1000).

[3] E. Balas and J. Xue, Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring, Algorithmica 15 (1996), 397-412.

[4] E. Balas and C. S. Yu, Finding a maximum clique in an arbitrary graph, SIAM Journal on Computing 15 (1986), 1054-1068.

[5] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, Linear programming and network flows, 3nd edition, John Wiley and Sons, New York, 2005.

[6] E. M. L. Beale, "Branch-and-bound methods for mathematical programming systems," Discrete Optimization, P. L. Hammer, E. L. Johnson, and B. H. Korte (Editors), Number 5 of Annals of Discrete Mathematics, North-Holland, 1979, pp. 201-219.

[7] E. M. L. Beale, Branch-and-bound methods for numerical optimization, M. M. Barritt and D. Wishart (Editors), COMPSTAT 80: Proceedings in Computational Statistics, 1980, pp. 11-20.

[8] E. M. L. Beale, "Integer programming," Computational Mathematical Programming, K. Schittkowski (Editor), Springer-Verlag, Berlin, 1985, pp. 1-24.

[9] E. M. L. Beale and J. A. Tomlin, Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables, Proceedings of the Fifth International Conference on Operations Research, J. Lawrence (Editor), 1970, pp. 447-454.

[10] N. Beaumont, An algorithm for disjunctive programming, European Journal of Operational Research 48 (1990), 362-371.

[11] P. Berman and A. Pelc, Distributed fault diagnosis for multiprocessor systems, Proceedings of the 20th Annual International Symposium on Fault-Tolerant Computing, Newcastle, UK, 1990, pp. 340-346.

[12] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The maximum clique problem," Handbook of Combinatorial Optimization, Kluwer Academic Publishers, Boston, MA,

1999, pp. 1-74.

[13] J. Bramel and D. Simchi-Levi, On the effectiveness of set covering formulations for the vehicle routing problem with time windows, Operations Research 45 (1997), 295-301.

[14] D. Brélaz, New methods to color the vertices of a graph, Communications of the ACM 22 (1979), 251-256.

[15] C. Bron and J. Kerbosch, Algorithm 457: Finding all cliques of an undirected graph, Communications of the ACM 16 (1973), 575-577.

[16] S. Burer, R. D. C. Monteiro, and Y. Zhang, Maximum stable set formulations and heuristics based on continuous optimization, Mathematical Programming Series A 94 (2002), 137-166.

[17] R. Carraghan and P. M. Pardalos, An exact algorithm for the maximum clique problem, Operations Research Letters 9 (1990), 375-382.

[18] K. Corradi and S. Szabo, A combinatorial approach for Keller's Conjecture, Periodica Mathematica Hungarica 21 (1990), 95-100.

[19] L. F. Escudero and S. Muñoz, On identifying dominant cliques, European Journal of Operational Research 149 (2003), 65-76.

[20] I. R. de Farias, E. L. Johnson, and G. L. Nemhauser, Branch-and-cut for combinatorial optimization without auxiliary 0-1 variables, Knowledge Engineering Review 16 (2001), 25-39.

[21] A. Frank, Some polynomial algorithms for certain graphs and hypergraphs, Proceedings of 5th British Combinatorial Conference, Winnipeg, Manitoba, Canada, 1975, pp. 211-226.

[22] E. J. Gardiner, P. J. Artymiuk, and P. Willett, Clique-detection algorithms for matching three-dimensional molecular structures, Journal of Molecular Graphics and Modeling 15 (1998), 245-253.

[23] M. Garey and D. Johnson, Computers and intractability, W. H. Freeman and Company, New York, 1979.

[24] H. M. Grindley, P. J. Artymiuk, D. W. Rice, and P. Willett, Identification of tertiary structure resemblance in proteins using a maximal common sub-graph isomorphism algorithm, Journal of Molecular Biology 229 (1993), 707-721.

[25] M. Grötschel, L. Lovász, and A. Schrijver, Geometric algorithms and combinatorial optimization, Springer-Verlag, Berlin, 1988.

[26] R. Horaud and T. Skordas, Stereo correspondence through feature grouping and maximal cliques, IEEE Transactions on Pattern Analysis and Machine Intelligence 11 (1989), 1168-

1180.

[27] K. Jansen, P. Scheffler, and G. Woeginger, The disjoint cliques problem, Operations Research 31 (1997), 45-66.

[28] D. Johnson and M. Trick (Editors), Cliques, coloring, and satisfiability: Second DIMACS implementation challenge, AMS, Providence, RI, 1996.

[29] G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1998), 359-392.

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

[31] G. Karypis and V. Kumar, Multilevel *k*-way partitioning scheme for irregular graphs, Journal of Parallel and Distributed Computing 48 (1998), 96-129.

[32] J. MacWilliams and N. J. A. Sloane, The theory of error correcting codes, North-Holland, Amsterdam, 1979.

[33] C. Mannino and A. Sassano, An exact algorithm for the maximum stable set problem, Computational Optimization and Applications 3 (1994), 243-258.

[34] A. Mehrotra and M. A. Trick, A column generation approach for graph coloring, INFORMS Journal on Computing 8 (1996), 344-354.

[35] E. M. Mitchell, P. J. Artymiuk, D. W. Rice, and P. Willet, Use of techniques derived from graph theory to compare secondary structure motifs in proteins, Journal of Molecular Biology 212 (1989), 151-166.

[36] G. L. Nemhauser and G. Sigismondi, A strong cutting plane/branch-and-bound algorithm for node packing, Journal of Operational Research Society 43 (1992), 443-457.

[37] G. L. Nemhauser and L. A. Wolsey, Integer and combinatorial programming, John Wiley and Sons, New York, 1988.

[38] P. R. J. Ostergard, A fast algorithm for the maximum clique problem, Discrete Applied Mathematics 120 (2002), 197-207.

[39] P. M. Pardalos and G. P. Rodgers, A branch and bound algorithm for the maximum clique problem, Computers and Operations Research 19 (1992), 363-375.

[40] F. Rossi and S. Smriglio, A branch-and-cut algorithm for the maximum cardinality stable set problem, Operations Research Letters 28 (2001), 63-74.

[41] E. C. Sewell, A branch and bound algorithm for the stability number of a sparse graph, INFORMS Journal on Computing 10 (1998), 438-447.

[42] A. M. Verweij, Selected applications of integer programming: a computational study, Ph.D. Thesis, University of Utrecht, Utrecht, Holland, September, 2000.

[43] B. Verweij and K. Aardal, An optimization algorithm for maximum independent set with applications in map labeling, Proceedings of the Seventh Annual European Symposium on Algorithms, Number 1643, J. Nesetril (Editor), in Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1999, pp. 426-437.

[44] S. de Vries and R. Vohra, Combinatorial auctions: a survey, INFORMS Journal on Computing 15 (2003), 284-309.

[45] D. West, Introduction to graph theory, Prentice-Hall, Upper Saddle River, New Jersey, 2001.

[46] W. E. Wilhelm, A technical review of column generation in integer programming, Optimization and Engineering 2 (2001), 159-200.

[47] J. Xue, Edge-maximal triangulated sub-graphs and heuristics for the maximum clique problem, Networks 24 (1994), 109-120.

Graph	$\|V\|$	$\|E\|$	Δ	P	$\|\hat{E}\|$	Method	RMP Rows	B&B Nodes	MP Sols.	B&P Time (sec.)
hamming8-2	256	1,024	3.1	20	626	(m1)	626	1	3	0.6
						(m2)	626	1	3	0.6
MANN_a9	45	72	7.3	5	26	(m1)	26	98	2,091	2.4
						(m2)	20	19	527	0.5
hamming6-2	64	192	9.5	8	96	(m1)	96	1	3	0.3
						(m2)	96	1	3	0.3
johnson8-4-4	70	560	23.2	3	240	(m1)	240	6	89	0.9
						(m2)	127	1	23	0.6
johnson16-2-4	120	1,680	23.5	8	1,082	(m1)	1,082	>3,500	>76,809	*
						(m2)	42	1	14	0.6
keller4	171	5,100	35.1	5	3,293	(m1)	3,293	21,067	405,073	4,557.1
						(m2)	1,995	12,523	307,029	1,812.2
hamming8-4	256	11,776	36.1	4	6,528	(m1)	6,528	1	6	4.4
						(m2)	3,707	1	12	6.1
brock200-3	200	7,852	39.5	2	3,463	(m1)	3,463	775	15,331	1,671.5
						(m2)	2,736	3,624	62,432	2,537.4
johnson8-2-4	28	168	44.4	8	137	(m1)	137	136	1,962	1.3
						(m2)	23	8	126	0.2
c-fat200-5	200	11,427	57.4	7	9,523	(m1)	9,523	45	1,321	86.1
						(m2)	9,393	33	1,238	86.3
p_hat300-1	300	33,917	75.6	2	16,580	(m1)	16,580	712	9,802	705.7
						(m2)	10,441	1,086	11,564	479.4
c-fat200-2	200	16,665	83.7	4	12,261	(m1)	12,261	17	233	20.9
						(m2)	11,406	8	127	19.2
c-fat200-1	200	18,366	92.3	2	8,999	(m1)	8,999	6	53	7.5
						(m2)	7,453	6	68	8.9

(m1) edge constraints only in master problem
(m2) clique constraints replace edge constraints in master problem
* exceeded memory capacity of 512 MB

Table 1: Comparison of methods (m1) and (m2)

Graph	$\|V\|$	$\|E\|$	Δ	P	Method	$\|\hat{E}\|$	RMP Rows	B&B Nodes	MP Sols.	B&P Time (sec.)
hamming8-2	256	1,024	3.1	20	(p1)	846	846	1	3	0.6
					(p2)	626	626	1	3	0.6
MANN_a9	45	72	7.3	5	(p1)	32	26	40	904	2.7
					(p2)	26	20	19	527	0.5
hamming6-2	64	192	9.5	8	(p1)	156	156	1	3	0.3
					(p2)	96	96	1	3	0.3
johnson8-4-4	70	560	23.2	3	(p1)	433	295	79	2,882	11.5
					(p2)	240	127	1	23	0.6
johnson16-2-4	120	1,680	23.5	8	(p1)	1,243	108	16	400	1.7
					(p2)	1,082	42	1	14	0.6
keller4	171	5,100	35.1	5	(p1)	4,499	3,226	>26,370	>853,215	*
					(p2)	3,293	1,995	12,523	307,029	1,812.2
hamming8-4	256	11,776	36.1	4	(p1)	10,650	7,434	>3,500	>38,126	*
					(p2)	6,528	3,707	1	12	6.1
brock200-3	200	7,852	39.5	2	(p1)	7,325	6,738	>9,334	>200,240	*
					(p2)	3,463	2,736	3,624	62,432	2,537.4
johnson8-2-4	28	168	44.4	8	(p1)	113	22	8	99	1.1
					(p2)	137	23	8	126	0.2
c-fat200-5	200	11,427	57.4	7	(p1)	11,201	11,077	>13	6,396	*
					(p2)	9,523	9,393	33	1,238	86.3
p_hat300-1	300	33,917	75.6	2	(p1)	31,511	24,833	>2,170	>84,660	*
					(p2)	16,580	10,441	1,086	11,564	479.4
c-fat200-2	200	16,665	83.7	4	(p1)	15,912	15,156	25	1,293	261.1
					(p2)	12,261	11,406	8	127	19.2
c-fat200-1	200	18,366	92.3	2	(p1)	16,696	14,504	90	2,047	158.5
					(p2)	8,999	7,453	6	68	8.9

(p1) partitioning the graph into chordal subgraphs
(p2) partitioning the graph using METIS
* exceeded memory capacity of 512 MB

Table 2: Comparison of methods (p1) and (p2)

Graph	$\|V\|$	$\|E\|$	Δ	P	$\|\hat{E}\|$	RMP Rows	Method	B&B Nodes	B&P Time (sec.)
hamming8-2	256	1,024	3.1	20	626	626	(b1)	1	0.7
						626	(b2)	1	0.6
MANN_a9	45	72	7.3	5	26	20	(b1)	13	0.6
						20	(b2)	19	0.5
hamming6-2	64	192	9.5	8	96	96	(b1)	1	0.2
						96	(b2)	1	0.3
johnson8-4-4	70	560	23.2	3	240	127	(b1)	1	0.6
						127	(b2)	1	0.6
johnson16-2-4	120	1,680	23.5	8	1,082	42	(b1)	*	*
						42	(b2)	1	0.6
keller4	171	5,100	35.1	5	3,293	1,995	(b1)	16,579	12,793.5
						1,995	(b2)	12,523	1,812.2
hamming8-4	256	11,776	36.1	4	6,528	3,707	(b1)	1	6.0
						3,707	(b2)	1	6.1
brock200-3	200	7,852	39.5	2	3,463	2,736	(b1)	>7,500	*
						2,736	(b2)	3,624	2,537.4
johnson8-2-4	28	168	44.4	8	137	23	(b1)	7	0.7
						23	(b2)	8	0.2
c-fat200-5	200	11,427	57.4	7	9,523	9,393	(b1)	51	293.2
						9,393	(b2)	33	86.3
p_hat300-1	300	33,917	75.6	2	16,580	10,441	(b1)	>2,000	*
						10,441	(b2)	1,086	479.4
c-fat200-2	200	16,665	83.7	4	12,261	11,406	(b1)	13	41.5
						11,406	(b2)	8	19.2
c-fat200-1	200	18,366	92.3	2	8,999	7,453	(b1)	9	9.5
						7,453	(b2)	6	8.9

(b1) branch on most fractional variable
(b2) branch on vertices (i.e., nodes) of a clique
* exceeded memory capacity of 512 MB

Table 3: Comparison of methods (b1) and (b2)

Graph	$\|V\|$	$\|E\|$	Δ	P	$\|\hat{E}\|$	RMP Rows	Z_{LP}	Z_H	Z_{IP}	B&B Nodes	MP Sols.	B&P Time (sec.)	CPAA Time (sec.)
hamming8-2	256	1,024	3.1	11	493	493	128.0	128	128	1	3	1.0	*
hamming8-2	256	1,024	3.1	20	626	626	128.0	128	128	1	3	0.6	*
hamming8-2	256	1,024	3.1	24	716	716	128.0	128	128	1	3	0.6	*
MANN_a9	45	72	7.3	5	26	20	18.0	16	16	19	527	0.5	620.8
MANN_a9	45	72	7.3	6	29	22	18.0	16	16	25	687	0.5	620.8
MANN_a9	45	72	7.3	8	35	31	18.5	16	16	43	1,363	0.7	620.8
hamming6-2	64	192	9.5	4	64	64	32.0	32	32	1	3	0.3	*
hamming6-2	64	192	9.5	6	114	114	32.0	32	32	1	3	0.3	*
hamming6-2	64	192	9.5	8	96	96	32.0	32	32	1	3	0.3	*
johnson8-4-4	70	560	23.2	2	140	62	14.0	14	14	1	22	1.7	14.9
johnson8-4-4	70	560	23.2	3	240	127	14.8	14	14	1	23	0.6	14.9
johnson8-4-4	70	560	23.2	6	348	217	16.4	14	14	13	492	0.8	14.9
johnson16-2-4	120	1,680	23.5	6	1,088	15	8.0	8	8	1	9	0.6	*
johnson16-2-4	120	1,680	23.5	8	1,082	42	8.5	8	8	1	14	0.6	*
johnson16-2-4	120	1,680	23.5	10	1,234	128	10.5	8	8	11	335	0.9	*

* exceeded memory capacity of 512 MB

Table 4: Instances taken from the Second DIMACS Implementation Challenge solved using the (m2)-(p2)-(b2) combination of methods

Graph	$\|V\|$	$\|E\|$	Δ	P	$\|\hat{E}\|$	RMP Rows	Z_{LP}	Z_H	Z_{IP}	B&B Nodes	MP Sols.	B&P Time (sec.)	CPAA Time (sec.)
keller4	171	5,100	35.1	4	3,003	1,853	17.8	8	11	14,456	329,556	1,934.2	3,075.4
keller4	171	5,100	35.1	5	3,293	1,995	18.1	8	11	12,523	307,029	1,812.2	3,075.4
keller4	171	5,100	35.1	8	3,744	2,554	20.7	8	11	24,690	694,846	12,831.5	3,075.4
hamming8-4	256	11,776	36.1	4	6,528	3,707	16.0	16	16	1	12	6.1	*
hamming8-4	256	11,776	36.1	5	7,707	4,505	20.8	16	16	440	21,373	536.5	*
hamming8-4	256	11,776	36.1	8	8,774	6,529	23.5	16	16	2,332	97,283	2,357.9	*
brock200-3	200	7,852	39.5	2	3,463	2,736	20.0	11	15	3,624	62,432	2,537.4	*
brock200-3	200	7,852	39.5	3	4,709	3,964	24.0	11	--	>10,024	>50,049	>14,400	*
johnson8-2-4	28	168	44.4	4	100	12	4.0	4	4	1	7	0.53	0.0
johnson8-2-4	28	168	44.4	5	124	32	5.3	4	4	6	95	0.45	0.0
johnson8-2-4	28	168	44.4	8	137	23	5.0	4	4	8	126	0.23	0.0

* exceeded memory capacity of 512 MB

Table 4: Instances taken from the Second DIMACS Implementation Challenge solved using the (m2)-(p2)-(b2) combination of methods

Graph	$\|V\|$	$\|E\|$	Δ	P	$\|\hat{E}\|$	RMP Rows	Z_{LP}	Z_H	Z_{IP}	B&B Nodes	MP Sols.	B&P Time (sec.)	CPAA Time (sec.)
c-fat200-5	200	11,427	57.4	4	8,118	7,985	66.7	58	58	33	1,328	157.8	41.4
c-fat200-5	200	11,427	57.4	7	9,523	9,393	66.7	58	58	33	1,238	86.3	41.4
c-fat200-5	200	11,427	57.4	8	9,787	9,655	66.7	58	58	33	1,599	112.1	41.4
p_hat300-1	300	33,917	75.6	2	16,580	10,441	12.9	5	8	1,086	11,564	479.4	3.9
p_hat300-1	300	33,917	75.6	3	21,972	13,968	16.0	5	8	4,032	44,928	2,302.3	3.9
p_hat300-1	300	33,917	75.6	5	26,448	17,813	20.7	5	8	8,834	122,254	6,411.4	3.9
c-fat200-2	200	16,665	83.7	4	12,261	11,406	26.2	24	24	8	127	19.2	1.2
c-fat200-2	200	16,665	83.7	5	13,156	12,300	25.5	24	24	8	138	22.6	1.2
c-fat200-2	200	16,665	83.7	8	14,504	13,783	26.9	24	24	26	685	56.6	1.2
c-fat200-1	200	18,366	92.3	2	8,999	7,453	13.0	12	12	6	68	8.9	1.0
c-fat200-1	200	18,366	92.3	3	12,137	9,996	14.0	12	12	19	219	19.4	1.0
c-fat200-1	200	18,366	92.3	6	15.232	12,807	13.3	12	12	13	181	24.6	1.0

* exceeded memory capacity of 512 MB

Table 4: Instances taken from the Second DIMACS Implementation Challenge solved using the (m2)-(p2)-(b2) combination of methods

π	$\lvert E \rvert$			$\lvert \hat{E} \rvert$			RMP Rows		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
0.01	37	62	50.0	0	3	0.4	0	3	0.4
0.05	219	276	246.2	67	107	91.4	67	107	86.6
0.10	461	534	497.2	224	281	252.7	195	243	220.5
0.15	704	780	742.6	390	453	419.2	332	397	361.8
0.20	949	1,040	983.1	549	629	591.6	462	538	503.2
0.30	1,405	1,530	1,471.9	883	977	939.0	747	819	786.4
0.40	1,919	2,089	1,981.6	1,268	1,407	1,321.4	1,025	1,143	1,079.4
0.50	2,395	2,586	2,475.8	1,622	1,779	1,697.2	1,231	1,362	1,325.2

Table 5: Randomly generated graphs

		Overall measures				B&P Run Times			CPAA Run Times		
π	M	Z^*_{LP} / Z^*_{IP}	Z_H / Z^*_{IP}	B&B Nodes	RMP Iterations	Min	Max	Avg	Min	Max	Avg
0.01	1	1.00	1.00	1.0	3.0	0.3	0.6	0.4	*	*	*
	20	1.00	0.91	1.0	3.0	0.2	0.5	0.3	*	*	*
	40	1.00	0.98	1.0	3.0	0.2	0.5	0.3	*	*	*
	60	1.00	0.98	1.0	3.0	0.2	0.5	0.3	*	*	*
	100	1.00	0.98	1.0	3.0	0.2	0.4	0.3	*	*	*
0.05	1	1.03	0.93	11.2	553.4	25.1	65.8	36.8	*	*	*
	20	1.01	0.94	6.2	350.8	3.5	27.9	10.2	*	*	*
	40	1.00	0.91	0.6	56.2	0.8	7.7	4.5	*	*	*
	60	1.00	0.93	1.2	74.6	0.3	7.3	3.3	*	*	*
	100	1.01	0.94	7.0	453.4	0.3	52.2	19.6	*	*	*
0.10	1	1.21	0.86	696.4	31,760.6	88.5	437.3	261.0	*	*	*
	20	1.06	0.92	22.6	1,348.6	13.8	46.3	27.1	*	*	*
	40	1.11	0.91	106.0	6,224.4	29.8	137.3	72.7	*	*	*
	60	1.10	0.93	72.8	4,459.2	27.7	97.7	55.3	*	*	*
	100	1.13	0.90	182.6	11,423.2	66.9	295.0	128.5	*	*	*
0.15	1	1.33	0.87	365.0	69,322.0	138.4	538.5	311.3	11,125.0	16,046.0	12,745.0
	20	1.24	0.91	384.6	17,263.0	29.7	135.9	84.1	1,688.6	5,281.4	3,100.2
	40	1.23	0.90	331.6	16,167.8	28.6	129.84	79.8	1,772.4	4,011.2	2,879.2
	60	1.24	0.89	247.2	11,392.4	44.0	76.9	59.2	1,992.0	4,626.9	2,998.8
	100	1.26	0.92	365.4	25,243.6	45.1	225.5	112.6	1,844.8	3,355.2	2,845.3
0.20	1	1.43	0.82	3,516.8	109,975.4	179.1	548.9	362.3	836.6	1,991.5	1,527.6
	20	1.30	0.90	403.8	15,594.6	35.7	83.2	63.9	245.4	1,032.8	570.6
	40	1.30	0.90	483.8	17,932.6	30.3	82.5	69.1	321.4	772.9	529.1
	60	1.31	0.91	447.0	17,645.2	46.3	83.3	67.4	212.6	1,141.9	437.8
	100	1.32	0.87	505.4	19,118.6	41.8	92.1	72.7	356.9	595.7	471.3
0.30	1	1.55	0.85	2,142.4	49,392.2	77.7	220.1	148.5	58.1	73.2	66.7
	20	1.43	0.88	351.2	9,881.2	25.2	45.2	34.4	15.2	30.3	20.6
	40	1.46	0.90	455.6	13,170.4	21.2	69.1	43.8	18.7	34.0	24.1
	60	1.46	0.86	687.8	18,199.4	38.7	122.9	59.6	19.6	40.9	28.7
	100	1.40	0.88	351.2	10,186.8	20.5	48.5	33.9	12.7	35.5	22.7

0.40	1	1.68	0.83	1,327.2	25,159.8	35.3	114.8	84.4	4.3	8.6	6.6
	20	1.57	0.86	444.6	9,500.2	19.5	43.9	33.8	2.5	3.7	3.1
	40	1.64	0.87	590.0	12,355.0	33.9	53.8	41.3	2.6	3.8	3.3
	60	1.51	0.87	353.2	7,780.8	19.3	37.1	27.2	3.2	4.4	3.8
	100	1.59	0.87	548.2	11,431.0	20.6	56.3	39.6	2.9	4.2	3.6
0.50	1	1.66	0.73	605.6	10,261.4	26.2	52.0	39.0	1.1	1.4	1.2
	20	1.68	0.78	377.0	6,637.4	18.2	35.7	25.1	0.7	1.0	0.8
	40	1.61	0.76	287.4	5,117.8	9.6	28.8	19.7	0.6	0.8	0.7
	60	1.56	0.75	227.6	4,100.2	12.9	20.1	16.3	0.6	0.8	0.7
	100	1.64	0.87	370.6	6,441.4	23.1	30.8	27.4	0.6	0.8	0.7

* exceeded memory capacity of 512 MB

Table 6: Test results on randomly generated graphs