APPLYING THE BOUNDARY POINT METHOD TO AN SDP RELAXATION OF THE MAXIMUM INDEPENDENT SET PROBLEM FOR A BRANCH AND BOUND ALGORITHM

by

Aaron T. Wilson

Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Science in Mathematics

with Specialization in Operations Research and Statistics

New Mexico Institute of Mining and Technology

Socorro, New Mexico

May, 2009

ABSTRACT

A common method, originally introduced by Lovász in 1979, for calculating an upper bound on the size of a maximum independent set for a graph is to consider a relaxation of the problem expressed as a semidefinite program (SDP). Today, the most prevalent method for solving a general SDP is with a primal-dual interior point method (IPM). These methods are highly developed, provide reliable convergence, and parallelize relatively well on a shared memory architecture. However, they are severely limited by their memory requirements, which grow with the square of the number of edges in the graph. Here, we investigate the boundary point method (BPM) developed by Povh, Rendl, and Weigele in 2006. Storage for this method grows as the square of the number of nodes in the graph, allowing us to bound much larger graphs. We have implemented the boundary point method in C within a branch-and-bound framework and discuss several methods used within that framework aimed at increasing the efficiency of the algorithm. We also compare the BPM with CSDP, an implementation of the IPM. Computational results show that the BPM is indeed useful for problems with a large ratio of edges to vertices, that performance can be improved within the branch-and-bound framework, and that it does not scale as well in a shared memory environment.

ACKNOWLEDGMENT

The author's greatest support came from Dr. Brian Borchers as his advisor, as head of his thesis committee, and as professor for many of his courses. The author's work in the mathematics department would not have been possible without the help and support of all his professors and fellow students. Additionally, he would like to thank Dr. Bill Stone and Dr. Rakhim Aitbayev for sitting on his thesis committee. Thanks also go to Richard Hahn for providing both a critical and supportive voice and to Maxx Kureczko for too many things to list here.

This thesis was typeset with LaTeX[1] by the author.

[1] LaTeX document preparation system was developed by Leslie Lamport as a special version of Donald Knuth's TeX program for computer typesetting. TeX is a trademark of the American Mathematical Society. The LaTeX macro package for the New Mexico Institute of Mining and Technology thesis format was adapted from Gerald Arnold's modification of the LaTeX macro package for The University of Texas at Austin by Khe-Sing The.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

This thesis is accepted on behalf of the faculty of the Institute by the following committee:

Brian Borchers, Advisor

Aaron T. Wilson Date

CHAPTER 1

SOLVING THE MAXIMUM INDEPENDENT SET PROBLEM USING BRANCH AND BOUND WITH A SEMIDEFINITE PROGRAM

1.1 Introduction

First, recall that a *graph* $G = (V, E)$ is defined by a finite set V of *vertices* and a set E of pairwise *edges* connecting one vertex to another. The ordered pair $(i, j) \in E$ then represents an edge from $i \in V$ to $j \in V$. In this paper we only consider undirected graphs, that is we make the assumption that $(i, j) \in E \Leftrightarrow (j, i) \in E$. Defining $|S|$ to represent the cardinality of the set S, let $n = |V|$ and $m = |E|$ unless otherwise stated.

Definition 1.1. An *independent set* on a graph $G = (V, E)$ is any $I \subseteq V$ such that $(i, j) \notin E$ for all $i, j \in I$. That is, I is a set of vertices whose elements are pairwise non-adjacent. Henceforth, we will use the term *neighbors* to denote two adjacent vertices and $neigh(i)$ to denote the set of all neighbors of vertex i.

Definition 1.2. A *maximal independent set* is any independent set $I \in V$ such that no more vertices in $V \backslash I$ may be added to I to form a larger independent set.

Definition 1.3. A *maximum independent set* $M \subseteq V$ on the graph $G = (V, E)$ is any independent set whose cardinality is greater than or equal to

the cardinality of all other independent sets. We will use $\alpha(G)$ to denote the cardinality of any maximum independent set on G.

Definition 1.4. Given $k \in \mathbb{Z}^+$ and a graph $G = (V, E)$, the *independent set problem* asks if there is an independent set with cardinality k on G.

The independent set problem is a well known example of an NP-hard problem, see [11]. Thus we know two things about this problem. First, given a subset of the vertices, we may check whether it is an independent set in polynomial time with respect to n. Second, there is no known algorithm that solves the independent set problem and runs in polynomial time with respect to n.

The corresponding optimization problem is the *maximum independent set* (MIS) problem,

$$\begin{array}{ll} \max & |I| \\ \text{s.t.} & I \text{ is an independent set on } G = (V, E). \end{array} \tag{1.1}$$

The MIS problem has numerous areas of application, including coding theory, fault tolerance, pattern recognition, and economics, amongst others. The Sloane graphs that will be discussed in Chapter 4 are designed such that an independent set on these graphs represents an error-correcting, binary code-set. An excellent discussion of some applications is presented by Pardalos and Xue in [24] and by Bomze, et al. in [4].

Not only is this a difficult problem to solve, there are no efficient algorithms for approximating the optimal value. Often, see [2, 4, 8, 9, 24, 26, 31], instead of attempting to approximate $\alpha(G)$ directly, relaxations of the

MIS problem which provide upper and lower bounds on $\alpha(G)$ are used within a branch-and-bound framework to solve the problem.

1.2 Branch and bound in general

A *branch-and-bound* algorithm is a type of problem solver often used for combinatorial maximization or minimization problems. It is a brute force search tree that is capable of cutting off some of its branches by taking advantage of known bounds on the objective value. We generalize the problem by assuming that we are considering a maximization problem of the following form.

$$\begin{array}{rl} \max & f(x) \\ \text{s.t.} & x \in S \end{array} \tag{1.2}$$

The branch-and-bound algorithm requires two things. First, a method for partitioning any domain of x, D, into k subsets $\{D_i\}_{i=1,\ldots,k}$ must exist. Second, there must be a method for bounding the optimal value of $f(x)$ from above and below for all $x \in S \cap D_i$.

A search tree is created by recursively partitioning the domain of the problem. If the upper bound found at some node in the search tree, with domain D_a, is found to be less than or equal to the lower bound at another node, with domain D_b, then there exists a feasible solution in D_b that has an objective value larger than any feasible solution in D_a. Therefore we may discard the node with domain D_a and all of its children from the search. Henceforth, when a node is removed from the search tree we will say that node has been "fathomed."

Algorithm 1 gives pseudocode for the recursive function *bandb*(T)

in the branch-and-bound algorithm. Here, it is assumed that some method, $upper(T)$, exists that is capable of bounding the maximum value of the objective function, f, from above on the domain T. The largest objective value known is used as a global lower bound on the optimal value, stored as *best*. That is

$$\exists x \in S \text{ s.t.} \qquad f(x) \geq best \tag{1.3}$$

$$f(x) \leq upper(T) \qquad \forall x \in S \cap T. \tag{1.4}$$

$bandb(T)$ returns an optimal solution within the given domain T. The branch-and-bound algorithm would consist of creating the initial domain D, initializing *best* with the best known lower bound, and calling $bandb(D)$. Note that in this

Algorithm 1 $bandb(T)$ returns $x \in T \cap S$ that maximizes f

Require: $best \leq f(x)\ \forall x \in S$

if $|T| == 1$ **then** {We've reached the bottom of the tree, return the only feasible solution}
  return T
else
  if $upper(T) > best$ **then**
    if $\exists \hat{x} \in T \cap S$ s.t. $f(\hat{x}) > best$ **then** {a better lower bound has been found }
      $best \leftarrow f(\hat{x})$
    end if
    partition T into k subsets, $\{T_i\}_{i=1,\ldots,k}$, such that $T_i \cap S \neq \emptyset$
    return

$$\underset{x \in \{bandb(T_i)\}_{i=1}^{k}}{\operatorname{argmax}} f(x)$$

  else
    return ignore
  end if
end if

algorithm, returning "ignore" should cause the algorithm to ignore the results

from that partition of the feasible region, hence fathoming that node in the tree. We could define $f(\text{ignore}) = -\infty$ to achieve this.

The efficiency of this algorithm is highly dependent on several yet undefined parts of the algorithm. The method of partitioning the feasible set T is discussed in §1.3. The heuristics used to find an initial lower bound and determine if there exists an $\hat{x} \in T$ that represents an independent set larger than *best* is discussed in §1.3.2. The method of finding the upper bound and the gap between the upper bound and optimal value are discussed in §1.4 and Chapter 2.

1.3 Branch and bound for the MIS problem

For the maximum independent set problem, the initial domain, D, is the set containing all subsets of the set of vertices in the graph, V; the feasible region, S, is the set of all independent sets on G; the objective function, f, is the cardinality of an independent set; and the lower bound, *best*, is provided by the cardinality of a known independent set. For now, we will leave the method for finding an upper bound undefined. This will be discussed in §1.4.

To solve this program using branch and bound, the feasible region is partitioned by branching on the decision to include or exclude individual vertices from the independent set. Let $I \subseteq V$ and $O \subseteq V$ denote the sets of vertices included and excluded from the independent set at some node in the search tree, respectively. We do not keep track of exactly what the feasible set of solutions is, given I and O. Instead, we exclude vertices from the graph so that the corresponding subgraph $\hat{G} = (V \backslash I \backslash O, E)$ cannot have an independent

set $\hat{I} \subseteq V \backslash I \backslash O$ where $\hat{I} \cup I$ is not an independent set on G. Excluding a vertex from the independent set is equivalent to simply removing it from the corresponding subgraph, as any independent set will exist somewhere else in the graph. Including a vertex in the independent set is equivalent to removing the vertex and all of its neighbors from the subgraph, as any independent set cannot include the neighbors of a vertex in the independent set, I.

Algorithm 2 shows Algorithm 1 modified for the maximum independent set problem. It is possible for this algorithm to return the empty set, but this would simply imply that no independent set larger than *best* was found. This is a useful property of the algorithm. By setting *best* to be larger than the largest known independent set, we can use the branch-and-bound tree to prove a bound on the largest independent set. Assuming that the algorithm does not return a larger independent set, we can be sure that there are none of this size in the graph.

We have left $upper(\hat{G})$ undefined here, but the simplest form this function might take is the number of vertices currently in the independent set plus the number of vertices still not set as in or out of the independent set, or

$$upper(\hat{G}) = |I| + |\hat{V}|. \tag{1.5}$$

Assuming that any form of *upper* that we choose will provide a better bound on the maximum size of the independent set, we can be sure that if we reach a leaf of the tree, then we have found an independent set larger than *best*. This can easily be seen by considering a node in the search tree that is the parent of a leaf. If adding one more vertex to the independent set will not create an

Algorithm 2 $mis\_bandb(\hat{G}, I)$ returns the largest independent set on $\hat{G} = (\hat{V}, \hat{E})$

Require: $best$ is the size of the largest known independent set and I is initialized as $\emptyset$

```
if |V̂| == 1 then {only one vertex in the subgraph, add it to I and return}
  if |I| + 1 > best then {a better lower bound has been found }
    best ← |I| + 1
  end if
  return V̂ ∪ I
end if
if upper(Ĝ) < best + 1 then {no better solutions can be found in this branch of the tree}
  return ∅
else
  compute a heuristic solution, Î within Ĝ let heur = |I| + |Î|
  if heur > best then {a better lower bound has been found }
    best ← heur
  end if
  pick a vertex i ∈ V̂ to branch on
  V_with ← mis_bandb( (V̂\{i}\neigh(i), Ê), I ∪ {i} )
  V_without ← mis_bandb( (V̂\{i}, Ê), I )
  if |V_with| > |V_without| then
    return V_with
  else
    return V_without
  end if
end if
```

independent set larger than *best* then the algorithm will not branch on that vertex.

1.3.1 Implementation of branch and bound

Several methods are used within the branch-and-bound algorithm in an attempt to minimize any unnecessary computation. By far, the most computationally expensive step of the branch-and-bound algorithm will be computing the upper bound. Thus we attempt to minimize the number of times that this bound is computed or decrease the size of the problem before computing the bound.

Consider a vertex in the unset subgraph for which each of its neighbors is connected to every other neighbor of that vertex. Now consider excluding that vertex from the independent set. Clearly we may still include only one of its neighbors in the independent set; doing so will exclude all of the neighbors of the original vertex and possibly several other vertices from the independent set. Including the vertex in the graph, however, excludes all of its neighbors and no other vertices. We need not consider excluding the vertex and all of its neighbors from the graph as this will have the same result on the unset subgraph, while including one less vertex in the independent set. Thus, before the bound is computed, a search of the unset subgraph is made. Any node whose neighbors are completely connected to each other is immediately set in the independent set, and the neighbors are set out.

Often times a computed bound will not be low enough to justify fathoming that node in the search tree and branching on some vertex will be

necessary. In this case, the bound computed becomes largely unnecessary. For this reason, we roughly estimate the value of the bound on the size of the independent set on the full graph and only bound when this estimate is smaller than the lower bound. This approximate value is decreased slightly each time a vertex is set out of the independent set and is updated each time a new bound is computed for the unset subgraph. The amount by which the estimate is decreased is constant and based on observation. Under the assumption that this is a relatively poor estimate of the bound and to prevent expanding the tree considerably without cause, every fifth generation in the tree is bounded regardless of the estimate of the next bound. The choice of five generations is based on observations with a small set of example problems. The choice in how we estimate the next bound and how often we force a bound have significant effects on the efficiency of the search tree. Further interest might be taken in developing other methods for making these choices.

To implement the branch-and-bound tree in a system where only one bound may be performed at a time, the order in which nodes are examined must be considered. A depth-first search might be helpful if no good lower bound was known, but this is rarely the case. Heuristics can often find the maximum independent set; the branch-and-bound tree is in this case used to confirm that these heuristic independent sets are indeed maximum. Heuristics used for finding an initial lower bound will be further discussed in §1.3.2. A depth-first search is used in our algorithm because it provides an easy framework for storing a solution to the formulation of the upper bound problem at a node in the search tree so that it may be used to increase the efficiency of a bound on a descendent of that node. This will be discussed in §3.3 and §3.4.

1.3.2 Heuristics

The branch-and-bound search tree is only pruned when an upper bound for a node is found to be smaller than the global lower bound, *best*, given by the largest independent set known. For this reason, the size of the search tree will be highly dependent on the value given to *best* when the algorithm is started (unless the simple heuristic completions of the solution within the algorithm happen to find a large independent set early in the search tree). Several very good heuristic algorithms exist which are capable of finding large independent sets on a given graph, see [4, 24, 27] for discussions of some of the search methods used for these heuristics. They are not used within the branch-and-bound tree because they are generally computationally expensive. Instead, we use them to attempt to find the largest independent set possible before starting the branch-and-bound algorithm.

Referring to algorithm 2, a heuristic solution is computed at each unfathomed node for comparison with the largest independent set known. The simple heuristic used here quickly finds a maximal independent set based on the partial solution I and the remaining subgraph $\hat{G}$. Unset vertices are added to the independent set at a node of the tree in no specific order until no more vertices may be added to the independent set. If a new, larger independent set is found, it is stored and its cardinality is used for the new lower bound.

1.4 Bounding the maximum independent set problem from above; The Lovász Theta Number

Numerous formulations of the maximum independent set problem have been introduced which provide an upper bound on $\alpha(G)$. For example,

the MIS problem can be formulated as a linear integer program,

$$\max \sum_{i=1}^{n} x_i \tag{1.6}$$

$$x_i + x_j \leq 1 \ \forall (i,j) \in E \tag{1.7}$$

$$x \in \{0,1\}^n. \tag{1.8}$$

Any optimal solution x to this program corresponds to a maximum independent set, I, on G where $i \in I$ if and only if $x_i = 1$. Relaxing the program by removing the integer constraint on x provides an upper bound on the problem that can be achieved using any number of linear programming methods. Similarly, the problem can be expressed as a quadratically constrained global optimization problem,

$$\max \sum_{i=1}^{n} x_i \tag{1.9}$$

$$x_i x_j = 0 \ \forall (i,j) \in E \tag{1.10}$$

$$x_i^2 = x_i, \ i = 1, ..., n \tag{1.11}$$

By introducing the quadratic constraint (1.11), any solution to this program will have $x \in 0, 1^n$ without the need for an explicit integer constraint. For large graphs, however, this non-convex global optimization problem becomes difficult to solve.

In 1979, Lozász defined the theta number of a graph, $\vartheta(G)$, to denote the minimum value of

$$\min_c \max_{1 \leq i \leq n} \frac{1}{(c^T u_i)^2} \tag{1.12}$$

$$|c| = 1 \tag{1.13}$$

over all orthonormal representations, $(u_1, ..., u_n)$, of a graph $G = (V, E)$. That is, over all systems of unit vectors $(u_1, ..., u_n)$ such that if $(i, j) \in E$ then $u_i^T u_j = 0$. In the same paper, see [18], he notes that $\vartheta(G)$ is an upper bound on both the Shannon capacity of the graph, $\Theta(G)$, and the size of a maximum independent set, $\alpha(G)$. Lovász later proved in [19] that the theta number can be calculated to arbitrary precision in polynomial time with respect to the size of the graph. Because the theta number is relatively easy to compute, it has been frequently used to bound the size of the maximum independent set problem, see [12, 13, 14, 25, 33].

Lovász proved in [19] that $\vartheta(G)$ is sandwiched between the size of a maximum independent set, $\alpha(G)$ – equal to the size of the largest clique on the inverse graph, $\omega(\bar{G})$ – and the number of colors needed to color the vertices of the inverse graph, $\chi(\bar{G})$.

$$\omega(\bar{G}) = \alpha(G) \leq \vartheta(G) \leq \chi(\bar{G}) \tag{1.14}$$

While we will use a bound provided by a semidefinite-programming formulation of $\vartheta(G)$ to bound the size of $\alpha(G)$, heuristically found minimal colorings have been used to bound this value as well. Tomita and Kameda, in [31], and Babel, in [2], present heuristic coloring algorithms used within a branch-and-bound framework to solve the maximum clique problem.

1.4.1 A semidefinite programming relaxation

László Lovász introduced $\vartheta(G)$ as an upper bound on $\alpha(G)$ in [18]. In the same paper he introduces the following SDP formulation of $\vartheta(G)$. We use J to denote a matrix of all ones, j to denote a column vector of all ones,

and $\langle A, B\rangle = \text{tr}(AB)$. It can be noted that $\langle J, X\rangle$ is simply the sum of the elements of X, whose optimal value is $\vartheta(G)$.

$$\begin{array}{rcll} \vartheta(G=(V,E)) & = & \max & \langle J, X\rangle \\ & & \text{s.t.} & \text{tr}(X) = 1 \\ & & & X_{ij} = 0 \ \ \forall (i,j) \in E \\ & & & X \succeq 0 \end{array} \tag{1.15}$$

It is relatively simple to show that this is a relaxation of (1.1). For any independent set $I \subseteq V$, consider $x \in \{0,1\}^n$, where $x_i = 1 \iff x \in I$. The matrix

$$X = \frac{xx^T}{j^T x} \tag{1.16}$$

is clearly feasible in (1.15), thus this SDP is a relaxation of the maximum independent set problem and we know that $\vartheta(G) \geq \alpha(G)$. While we do not consider a better bound than that given by $\vartheta(G)$, this bound can be tightened by adding additional constraints to (1.15). Schrijver's number, introduced in [28], is attained by adding non-negativity constraints, $X_{ij} \geq 0$. Lovász and Shriver additionally introduce several inequality constraints that potentially lower the bound even further in [20]. Computational results examining both of these modifications are presented by Dukanovic and Rendl in [7].

1.4.2 A general SDP and its dual

So that we can discuss some of the methods for solving an SDP, we consider the general semidefinite program of the form

$$\begin{array}{rll} \max & \langle C, X\rangle & \\ \text{s.t.} & A(X) = b & (P) \\ & X \succeq 0, & \end{array}$$

where we define

$$[A(X)]_i = \langle A_i, X\rangle \text{ for } i = 1, ..., m \tag{1.17}$$

and we have $X, C, A_i \in \mathbb{S}^n$ and $b \in \mathbb{R}^m$. Here m and n simply represent the number of equality constraints and the dimensions of X, respectively. We note that (1.15) can easily be expressed in this form. When it is, note that $C = J$, and b is a vector of m zeros and a single one (corresponding to the constraint $\mathrm{tr}(X) = 1$).

Introducing Lagrangian multipliers y and Z for the equality constraints and semidefinite constraint respectively, the Lagrangian of the equivalent minimization problem is

$$L(X; Z, y) = -\langle C, X\rangle - \langle Z, X\rangle + y^T(A(X) - b) \tag{1.18}$$

$$= \left\langle -C - Z + \sum_{i=1}^{m} y_i A_i, X \right\rangle - b^T y \tag{1.19}$$

$$= \left\langle -C - Z + A^T(y), X \right\rangle - b^T y, \tag{1.20}$$

where we define

$$A^T(y) = \sum_{i=1}^{m} y_i A_i \tag{1.21}$$

The objective function of the Lagrangian dual is obtained by taking the infimum of the Lagrangian over all $X \in \mathbb{S}^n$. This infimum is clearly unbounded unless

$$-C - Z + A^T(y) = 0. \tag{1.22}$$

Thus, the minimization form of the Lagrangian dual of (P) is

$$\begin{array}{ll} \min & b^T y \\ \text{s.t.} & A^T(y) - C = Z \\ & Z \succeq 0. \end{array} \tag{D}$$

Note here that the dual objective value for the SDP form of the Lovász theta number is the single y value corresponding to the $\mathrm{tr}(X) = 1$ constraint.

If there is a strictly feasible point X for the primal form (P) then strong duality holds by Slater's condition [5]. The necessary and sufficient KKT conditions for optimality are then

$$X \succeq 0,\ A(X) = b,\ Z \succeq 0,\ A^T(y) - Z = C,\ \langle Z, X \rangle = 0. \tag{1.23}$$

We can note that the SDP form of the Lovász theta number, (1.15), has such a strictly feasible solution. For example, $X = \frac{1}{n}I$ (where I is the nxn identity matrix) satisfies

$$\mathrm{tr}(X) = 1 \tag{1.24}$$

$$X_{ij} = 0 \quad \forall (i,j) \in E \tag{1.25}$$

$$X \succ 0. \tag{1.26}$$

We will be solving this problem numerically, so we only expect to get feasible solutions within some tolerance of optimality. A feasible solution for (1.15) will provide an objective value that is a lower bound on the upper bound given by $\vartheta(G)$ so we consider the dual of the SDP, a minimization program. By weak duality, any dual feasible solution will provide an objective value that is an upper bound on the upper bound given by $\vartheta(G)$.

1.5 Interior point methods

Presently, one of the most common ways of solving an SDP is with interior point methods. Interior point methods have been heavily developed in the past two decades and a wealth of information on the subject can be found, [1, 6, 10, 22] provide a good overview of some of these developments.

1.5.1 Solving an SDP with the interior point method

Any optimal solution to (P) and (D) will satisfy the KKT conditions given in (1.23). The interior point method is an iterative method that updates a primal-dual feasible solution, $(X; y, Z)$, by first computing a Newton's method step toward satisfying a modified version of the equalities in the KKT conditions, then truncating that step to ensure that the updated X and Z remain positive semidefinite.

The modification to the KKT conditions is made because the constraint $\langle X, Z\rangle = 0$, or equivalently $XZ = 0$ becomes poorly conditioned as X or Z approach the boundary of positive semidefinite matrices. To push the solution away from this boundary until solutions begin to approach optimality, the constraint is replaced by $XZ = \nu I$ for some $\nu > 0$. This ensures that X and Z are both strictly positive definite. In the limit as $\nu \to 0$ the solutions to the new constraints

$$A(X) = b \tag{1.27}$$

$$A^T(y) - Z = C \tag{1.28}$$

$$XZ = \nu I. \tag{1.29}$$

where $X, Z \succeq 0$ approach the optimal solution to (P) and (D). In fact, it can be shown that the exact solutions to these equations, $(X(\nu); y(\nu), Z(\nu))$, form a differentiable path through the feasible space for the primal-dual pair. This is called the *central path*.

Given that $(X; y, Z)$ are primal and dual feasible, an updated feasible

solution, $(X + \Delta X; y + \Delta y, Z + \Delta Z)$, should satisfy

$$A(\Delta X) = 0 \tag{1.30}$$

$$A^T(\Delta y) - (\Delta Z) = 0 \tag{1.31}$$

$$X\Delta Z + \Delta X Z = -XZ + \nu I \tag{1.32}$$

if we eliminate the second order term $\Delta X \Delta Z$. Unfortunately, solving this system for $(\Delta X; \Delta y, \Delta Z)$ may yield non-symmetric updates to X and Z. Several classes of interior point methods arise from the choice of how to force the solutions of this system to be symmetric. For example, Alizadeh, Haeberly, and Overton [1] propose symmetrizing the nonsymmetric comstraint (1.29) as

$$\frac{1}{2}(XZ + ZX) = \nu I. \tag{1.33}$$

In §1.5.2 we briefly discuss an implementation of the interior point method that will be used for comparison with the algorithm we used. In Chapter 2 we will review the development of the algorithm we use, a boundary point method originally presented in [25] that allows us to consider problems of considerably larger size.

1.5.2 CSDP; An interior point method implementation for comparison

CSDP is a semidefinite programming library written in C, created by Brian Borchers. In this algorithm, the equation (1.29) is not symmetrized before updating X. Instead, the update to X is symmetrized by averaging the off-diagonal elements.

$$\Delta X = \frac{\Delta X + \Delta X^T}{2}$$

The algorithm performs what is equivalent to solving the original KKT conditions ((1.30, 1.31, 1.32) with $\nu = 0$) then projecting that solution onto the central path.

Note that if a solution is on the central path, then (1.29) is satisfied and

$$\nu = \frac{\text{tr}(XZ)}{n} \tag{1.34}$$

We define

$$\mu(X, y, Z) = \frac{\text{tr}(XZ)}{n} \tag{1.35}$$

It has been shown, see [15], that projecting a feasible solution onto the central path results in a new feasible solution with the same value of μ. Solving the system of equations (1.30, 1.31, 1.32) for (X, y, Z) with $\nu = 0$ (that is, solving the original KKT conditions) provides updates that produce a feasible solution, $(\hat{X}, \hat{y}, \hat{Z})$, that will not necessarily be on the central path. Rather than attempting to project the solution $(\hat{X}, \hat{y}, \hat{Z})$ onto the central path, the system (1.30, 1.31, 1.32) is solved using the original (X, y, Z) and $\nu = \mu(\hat{X}, \hat{y}, \hat{Z})$. This will produce the same result without having to consider the possibly poorly-conditioned system at $(\hat{X}, \hat{y}, \hat{Z})$.

It has been shown, see [15], that solving the system of equations (1.30, 1.31, 1.32) is equivalent to solving

$$[A(Z^{-1}A^T(e_1)X) \dots A(Z^{-1}A^T(e_m)X)]\Delta y \equiv M\Delta y = -b \tag{1.36}$$

$$\Delta X = -X - \nu Z^{-1} - X(A^T(\Delta y))Z^{-1} \tag{1.37}$$

$$\Delta Z = A^T(\Delta y) \tag{1.38}$$

This is achieved using a Cholesky factorization of the matrix M and Z, which is the most computationally intensive part of the algorithm. Factoring the $m+1$ by $m+1$ matrix M for the maximum independent set problem will usually consume considerably more time than factoring an n by n matrix as we almost always have considerably more edges (m) than vertices (n) in the graph. So a singe iteration of the algorithm takes $O(m^3)$ time. Once $(\Delta X, \Delta y, \Delta Z)$ is computed, a line search is performed and (X, y, Z) is updated

$$(X, Y, Z) = (X + \alpha_p \Delta X, y + \alpha_d \Delta y, Z + \alpha_d \Delta Z) \tag{1.39}$$

$$\text{s.t.} \quad X + \alpha_p \Delta X \succeq 0 \tag{1.40}$$

$$Z + \alpha_d \Delta Z \succeq 0 \tag{1.41}$$

This brief overview of CSDP is certainly not complete. Numerical round-off error is assumed and solutions which are not strictly feasible are accounted for within the algorithm. The convergence criteria can be specified by the user, but by default are given by

$$\frac{\langle X, J \rangle - b^T y}{1 + |\langle X, J \rangle| + |b^T y|} \leq \tau_{gap} \tag{1.42}$$

$$\frac{||A(X) - b||}{1 + |b|} \leq \tau_{primal} \tag{1.43}$$

$$\frac{||Z + C - A^t(y)_F||_F}{1 + ||C||_F} \leq \tau_{dual}. \tag{1.44}$$

CHAPTER 2

SOLVING A GENERAL SDP WITH THE BOUNDARY POINT METHOD

Here, we review the boundary point method introduced in [25] that finds a solution to a general SDP with only equality constraints.

2.1 The augmented Lagrangian

We consider the dual formulation of the SDP and introduce Lagrangian multiplier $X \in \mathbb{S}^n$ for the equality constraint and augment the partial Lagrangian of the dual program (D) as follows.

$$L(y, Z; X; \sigma) = b^T y + \langle X, Z + C - A^T(y) \rangle + \frac{\sigma}{2} \left|\left| Z + C - A^T(y) \right|\right|_F^2 \quad (2.1)$$

$$= b^T y + \frac{\sigma}{2} \left|\left| Z + C - A^T(y) + \frac{1}{\sigma} X \right|\right|_F^2 - \frac{1}{2\sigma} ||X||_F^2 , \quad (2.2)$$

where $\sigma > 0$ and we are using the Frobenius matrix norm, defined by the component-wise product discussed earlier

$$||B||_F^2 = \langle B, B \rangle. \quad (2.3)$$

We have chosen X as the Lagrange multiplier because we will be able to show that the optimal X^* satisfies the necessary and sufficient KKT conditions for the optimality of the primal-dual pair, (1.23).

If we minimize this function over y and $Z \succeq 0$, the augmentation of the Lagrangian will penalize solutions that violate the equality constraints

according to the square of the norm of the residual scaled by σ. As σ is increased, the penalty is increased and the optimum values are brought closer to a minimum of the Lagrangian that satisfies the equality constraints. The augmented Lagrangian approach involves iteratively minimizing (2.1) with respect to y and $Z \succeq 0$ while holding X constant, updating the Lagrange multipliers, X, then increasing σ. This is repeated until the KKT conditions, (1.23), are satisfied [3].

We do not include a Lagrangian multiplier for Z, nor do we penalize solutions that do not satisfy the positive semidefinite constraint because this boundary point method will explicitly solve for Z that minimizes the partial Lagrangian subject to $Z \succeq 0$.

Guaranteeing convergence can be complicated and is not addressed thoroughly here. In general, we will ensure convergence if σ is not decreased at any iteration and the Lagrangian is minimized at the k^{th} iteration such that

$$\left|\left|(\nabla_y \hat{L})(y^k, Z^k)\right|\right|_2 + \left|\left|(\nabla_Z \hat{L})(y^k, Z^k)\right|\right|_F \leq \epsilon_k, \tag{2.4}$$

where $\hat{L}$ is the full Lagrangian, including Lagrange multipliers for the positive semidefinite constraint $Z \succeq 0$, and $\{\epsilon_k\}$ are chosen carefully, e.g. $\sum_{k=1}^{\infty} \epsilon_k < \infty$. See [3] for more details.

Minimizing the augmented Lagrangian with respect to $Z \succeq 0$ and y does not provide a solution where the equality constraints are satisfied. Instead, after the k^{th} minimization of the Lagrangian, it provides a solution where, according to [23],

$$Z^k + C - A^T(y^k) \approx -\frac{1}{\sigma}(X^* - X^{k-1}), \tag{2.5}$$

where we have used * to denote the optimal value of variables. To update the Lagrange multipliers, X, the method takes advantage of (2.5) under the assumption that we want $X^k \approx X^*$ yielding

$$X^k \leftarrow X^{k-1} + \sigma(Z^k + C - A^T(y^k)). \tag{2.6}$$

2.1.1 Minimizing the augmented Lagrangian

Minimizing $L(y, Z; X; \sigma)$ to find $Z \succeq 0$ and y while holding X constant is equivalent to the program

$$\min \quad L(y, Z) = b^T y + \frac{\sigma}{2} \left|\left| Z + C - A^T(y) + \frac{1}{\sigma} X \right|\right|_F^2 \tag{2.7}$$

$$\text{s.t.} \quad Z \succeq 0. \tag{2.8}$$

Following the method used in [25], we will use a coordinate descent method (for which convergence is guaranteed) to solve this convex program.

Holding Z constant, we can minimize (2.7) with respect to y, which is unconstrained, by solving $\nabla_y L = 0$ explicitly. Taking advantage of

$$[\nabla_y \langle A^T(y), B\rangle]_i = \frac{\partial}{\partial y_i} \left(\sum_{j=1}^{m} y_j \langle A_j, B\rangle \right) \tag{2.9}$$

$$= \langle A_i, B\rangle \tag{2.10}$$

$$\nabla_y \langle A^T(y), B\rangle = A(B), \tag{2.11}$$

we can write

$$\nabla_y L = b - \sigma A(Z + C - A^T(y) + \frac{1}{\sigma} X) = 0. \tag{2.12}$$

Isolating y yields the following linear system.

$$A(A^T(y)) = A(Z + C + \frac{1}{\sigma} X) - \frac{1}{\sigma} b \tag{2.13}$$

If we hold y constant, minimizing (2.7) with respect to $Z \succeq 0$ becomes

$$Z = \text{argmin}_{Y \succeq 0} \left|\left|Y - W(y)\right|\right|_F , \tag{2.14}$$

where we have defined $W(y) = A^T(y) - C - \frac{1}{\sigma}X$. Thus Z is the projection of $W(y)$ onto the cone of positive semidefinite matrices, denoted

$$Z = W(y)_+ \succeq 0 \tag{2.15}$$

Vandenberghe and Boyd show that this can be calculated from the eigenvalue decomposition of $W(y)$ in [5]. If $W = Q\Lambda Q^T$ is the eigenvalue decomposition of W and $\Lambda = \Lambda_+ + \Lambda_-$ and $Q = Q_+ + Q_-$ are a partitioning of the eigenvalue and eigenvector matrices into columns corresponding to positive and negative eigenvalues then

$$W_+ = Q_+\Lambda_+Q_+^T \succeq 0 \tag{2.16}$$

By repeatedly minimizing this convex problem with respect to one variable and its constraints, then the other and its constraints, i.e. cyclically updating y and Z according to

$$A(A^T(y)) = A(Z + C + \frac{1}{\sigma}X) - \frac{1}{\sigma}b \tag{2.17}$$

$$Z = W(y)_+ \tag{2.18}$$

the pair (y, Z) will converge to the minimizers of the dual Lagrangian. The optimum values will satisfy both of these equations simultaneously.

2.1.2 Convergence of the minimization of the augmented Lagrangian

In §2.1 we noted that part of guaranteeing convergence required that, at the k^{th} iteration of the augmented Lagrangian method, the approximate

optimal values of the minimization of the Lagrangian, (y^k, Z^k), satisfy

$$\left|\left|(\nabla_y \hat{L})(y^k, Z^k)\right|\right|_2 + \left|\left|(\nabla_Z \hat{L})(y^k, Z^k)\right|\right|_F \leq \epsilon_k, \tag{2.19}$$

where $\hat{L}$ is the full Lagrangian, including Lagrange multipliers for the positive semidefinite constraint $Z \succeq 0$, and $\{\epsilon_k\}$ are chosen carefully (see [3]) and at least satisfy $\epsilon_k \to 0$. After Z^k is updated in the coordinate descent method described above, we know that $(\nabla_Z \hat{L})(y^k, Z^k) = 0$. Because the semidefinite constraint does not involve y, we know that $\nabla_y \hat{L} = \nabla_y L$. Therefore the condition for convergence should be

$$\left|\left|(\nabla_y L)(y^k, Z^k)\right|\right|_2 = \left|\left|b - \sigma A(Z^k + C - A^T(y^k) + \frac{1}{\sigma}X)\right|\right|_2 \leq \epsilon_k \tag{2.20}$$

If we define

$$W_- = W - W_+ = Q_- \Lambda_- Q_-^T \preceq 0 \tag{2.21}$$

Then we can note that

$$A^T(y) - Z - C - \frac{1}{\sigma}X = W(y) - Z \tag{2.22}$$

$$= W(y) - W(y)_+ \tag{2.23}$$

$$A^T(y) - Z - C - \frac{1}{\sigma}X = W(y)_- \tag{2.24}$$

so we can rewrite the convergence criterion as

$$||b - A(-\sigma W(y)_-||_2 \leq \epsilon_k \tag{2.25}$$

and note that this is simply a measure of the primal feasibility of $-\sigma W(y)_-$.

2.2 Updating Lagrange multipliers; an algorithm

Now that we have established a method for minimizing the augmented Lagrangian at one step in the process, we can consider the update of X given by (2.6) after computing the approximate minimizers of the Lagrangian, (y^k, Z^k)

$$X^k \leftarrow X^{k-1} + \sigma(Z^k + C - A^T(y^k)) \tag{2.26}$$

$$= -\sigma W(y^k)_- \tag{2.27}$$

where we have taken advantage of (2.24). Note that the stopping criterion for the minimization of the augmented Lagrangian is equivalent to achieving approximate primal feasibility for the updated X. Therefore ϵ_k is a bound on the 2-norm of the residual of the primal constraints.

Note also that at each iteration, if we observe that $Q_-^T Q_+$ is zero because the eigenvectors are linearly independent,

$$X^k = -\sigma W(y^k)_- \succeq 0 \tag{2.28}$$

$$Z^k = W(y^k)_+ \succeq 0 \tag{2.29}$$

$$\langle X^k, Z^k \rangle = -\sigma \mathrm{tr}(Q_- \Lambda_- Q_-^T Q_+ \Lambda_+ Q_+^T) = 0. \tag{2.30}$$

so all of the KKT conditions for the primal-dual pair given in (1.23) are satisfied except primal and dual feasibility. Thus a natural stopping criterion for an algorithm based on the augmented Lagrangian method would involve setting tolerances for the primal and dual infeasibility.

Algorithm 3 shows the boundary point method where we have let τ_d and τ_p denote the desired bounds on the matrix norm and 2-norm of the residuals of primal and dual feasibility, respectively. This algorithm is called

the boundary point method because the variable Z always remains on the boundary of the cone of positive semidefinite matrices.

Algorithm 3 The boundary point method

Choose $\sigma > 0$
$k \leftarrow 0$
Let $X^k, Z^k = 0$
while $\delta_d > \tau_d$, $\delta_p > \tau_p$ **do**
  $k \leftarrow k + 1$
  while $\delta_p > \epsilon_k$ **do**
    Solve for y^k: $A(A^T(y^k)) = A(Z^{k-1} + C + \frac{1}{\sigma}X^{k-1}) - \frac{1}{\sigma}b$
    $W \leftarrow A^T(y^k) - C - \frac{1}{\sigma}X^{k-1}$
    $Z^k \leftarrow W_+$
    $M \leftarrow -\sigma W_-$
    $\delta_p = ||A(M) - b||_2$
  end while
  $X^k \leftarrow M$
  $\delta_d \leftarrow \left|\left|Z^k - A^T(y^k) + C\right|\right|_F$
  choose $\epsilon^{k+1} \leq \epsilon^k$
  increase σ if necessary
end while

2.3 The boundary point method for the maximum independent set problem

The two most computationally intensive operations in the boundary point method are updating y^k by solving the linear system (2.17) and performing the eigenvalue decomposition needed to update Z^k (2.18), and later X^k. Even for a general SDP, solving the linear system can be done relatively easily, as the matrix form of the linear operator $A(A^T(\cdot))$ is constant and positive definite. One might perform a Cholesky factorization of the matrix form once, using this to solve for y^k each time. Thus the solution for y^k grows, at worst,

as m^2. Note here that the matrix form of $A(A^T(\cdot))$ is in S_+^n and may be dense for an arbitrary SDP. This may introduce constraints on the size of solvable problems due to memory constraints as m grows.

When we apply the boundary point method to the maximum independent set problem, solving the linear system for y^k becomes a very simple operation as $A(A^T(y^k))$ can be represented an elementwise product with y^k. The constraints $\langle A_i, X\rangle = b_i$ for $i = 1, ..., n$ are used to enforce the equality constraints given in (1.15), specifically $\mathrm{tr}(X) = 1$ and $X_{ij} = 0$ for all $(i, j) \in E$. Clearly the matrixes $\{A_i\}_{i=1}^n$ share no common nonzero entries so $\langle A_i, A_j\rangle = 0$ for all $i \neq j$ and thus

$$\left[A(A^T(y))\right]_i = \left[A\left(\sum_{k=1}^{n} y_k A_k\right)\right]_i \tag{2.31}$$

$$= \sum_{k=1}^{n} y_k \langle A_i, A_k\rangle \tag{2.32}$$

$$= y_i \langle A_i, A_i\rangle \tag{2.33}$$

It is important to note that because the matrix form of $A(A^T(\cdot))$ is diagonal, the storage requirements for this problem are considerably less than those for the interior point method and CSDP. Solving the system of equations (2.17) becomes an order m operation. The only intensive computation remaining within a single iteration is the eigenvalue decomposition which grows, at worst, as n^3. Compare this to CSDP, the interior point method discussed in Section 1.5.2, in which the time for a single iteration grew as m^3, where m is usually much larger than n for any graph. This difference in time for individual iterations is not as important as it might seem, as often CSDP converges faster than the boundary point method.

Chapter 3 discusses the implementation of the boundary point method within the branch-and-bound algorithm in C. All of the methods described therein are aimed at decreasing the time for an eigenvalue decomposition, at reducing the size of the matrix for the eigenvalue decomposition, or reducing the number of eigenvalue decompositions that will be computed.

CHAPTER 3

IMPLEMENTATION OF THE BOUNDARY POINT METHOD FOR THE MIS PROBLEM

Algorithm 3 is based on the augmented Lagrangian method, where careful choices of ϵ^k and σ can guarantee convergence. In practice, however, such stringent requirements are rarely necessary. Updating X requires very little computation once the eigenvalue decomposition is complete. If we assume that an update to X is closer to optimality after only one iteration of the coordinate descent method then, rather than performing several expensive updates of the (y, Z) pair to achieve a norm of the residual of the primal constraints less than ϵ^k, we choose to update X after each update of y and Z. This eliminates the inner loop of the algorithm entirely, while convergence is still achieved, in practice, for the maximum independent set problem. Some general SDP problems do not converge, in practice, with this method. We have not tested general SDP problems extensively, though.

3.1 Forcing dual feasibility

In order for the dual objective value to be a valid bound on the size of the maximum independent set, the pair (y, Z) must be strictly dual feasible. To achieve this we can compute Z directly from y using

$$Z = A^T(y) - J. \tag{3.1}$$

so that the dual equality constraint is satisfied to within the numerical accuracy of the computer. Then, to check whether Z is semidefinite, the smallest eigenvalue of Z can be computed. If it is non-negative, then the dual constraints are satisfied and the dual objective, the element of y corresponding to the primal $\mathrm{tr}(X) = 1$ constraint (let us denote it by y_{tr}), is a bound on the size of the maximum independent set. If the smallest eigenvalue is negative, let us denote it by $-\alpha^2$, we can easily adjust Z to be semidefinite using

$$Z = Z + \alpha^2 I. \tag{3.2}$$

To maintain the dual equality constraint we must also increase y_{tr} by α^2. This results in an increase to the dual objective value of α^2. These steps are performed at the end of the BPM algorithm to give a valid bound.

Within the BPM algorithm, we are not concerned with exactly what this feasible dual objective value is, rather we are concerned with whether that value is low enough to fathom the current node in the branch-and-bound tree. If the current (possible infeasible) dual objective value is low enough to fathom the current node in the branch-and-bound tree (that is, less than the cardinality of the largest known independent set, *best*, plus one) we want to check for a strictly feasible solution that will fathom the node. Z is computed from y with y_{tr} set to $best + 0.99$, providing a (y, Z) pair based on the current solution that, if feasible, will fathom the current node. Checking feasibility requires only checking that this new Z is positive semidefinite, which is achieved by attempting to perform a Cholesky decomposition of Z, which will fail if the matrix is not positive definite.

3.2 The eigenvalue decomposition, parallelization

All of the computations performed in algorithm 3 are relatively simple compared to the eigenvalue decomposition. LAPACK is a linear algebra package written in Fortran77 that provides, among others, routines that perform eigenvalue decompositions on symmetric matrices. Specifically, these are DSYEV, DSYEVD, DSYEVR, and DSYEVX. All of them are based as much as possible on the low-level linear algebra routines given by BLAS (Basic Linear Algebra Subprograms). The boundary point method requires the projection of a matrix, $W(y)$, onto the cone of positive semidefinite matrices, denoted $W(y)_+$. As was mentioned before, if $W = Q\Lambda Q^T$ is the eigenvalue decomposition of W and $\Lambda = \Lambda_+ + \Lambda_-$ and $Q = Q_+ + Q_-$ are a partitioning of the eigenvalue and eigenvector matrices into columns corresponding to positive and negative eigenvalues then

$$W_+ = Q_+\Lambda_+Q_+^T \tag{3.3}$$

or, taking advantage of $W = W_+ + W_-$,

$$W_+ = W - W_- = W - Q_-\Lambda_-Q_-^T \tag{3.4}$$

Thus to find W_+ we need either all of the positive eigenvalue-eigenvector pairs, or all of the negative ones. For this reason, we chose to use one of the two LAPACK methods, DSYEVX and DSYEVR, that take an interval of real numbers and only return the eigenvalue-eigenvector pairs with eigenvalues in that interval. Thus we can easily compute only the positive eigenvalues or the negative eigenvalues to reduce computational time. To decide which, the BPM implementation stores the number of positive eigenvalues found from the last

decomposition of $W(y)$. Under the assumption that $W(y)$ and its eigenvalues do not drastically change from iteration to iteration, which should be the case as we approach optimality, we compute the positive eigenvalues if the last decomposition had more than half negative eigenvalues and vice versa.

Kozin and Deegan investigated the performance of these routines on multiple-core architectures in [17]. It was noted that many of the algorithms do not parallelize well because they are heavily dependent on matrix-vector multiplication, which cannot be parallelized well because of memory bandwidth issues. Matrix-matrix multiplications, on the other hand, can be efficiently parallelized. Because our test machine has a multiple core architecture and DSYEVR is more dependent on matrix-matrix multiplications than DSYEVX, we chose to use the former for our eigenvalue decompositions.

3.3 Fathoming with previous solutions

In order to fathom a node in the tree (remove its children from the tree), we need a feasible solution to the dual problem (D) that has an objective value less than the size of the largest known independent set. Feasibility is defined by the two constraints in the general SDP

$$A^T(y) - C = Z \tag{3.5}$$

$$Z \succeq 0 \tag{3.6}$$

Note that there are no restrictions on y, so given values of y we could calculate Z and test to see if it is positive semidefinite. If Z is PSD and the objective value, $b^T y$ in the general SDP, is smaller than the largest independent set we can fathom the node without having to use the BPM or CSDP method. Most

of the computational cost for this check will be in calculating the smallest eigenvalue of the resulting Z but this is small in comparison with the cost of the eigenvalue decompositions necessary for the BPM method or the Cholesky factorization for CSDP.

The general SDP expresses its equality constraints as

$$\langle A_i, X \rangle = b_i \tag{3.7}$$

When we consider the SDP form of the Lovász theta problem, we can note that $2m$ of the constraints are given by $X_{ij} = 0$ for all $(i, j) \in E$ (this gives $2m$ constraints because each edge is bidirectional). So we can say that A_k is a matrix of all zeros except for the (i, j) entry, where (i, j) is the k^{th} edge in an arbitrary ordering of the edges (including the duplicate reversed edges). Additionally, $b_k = 0$ for $k = 1, ..., 2m$. The final constraint is given by $diag(X) = 1$. We can say that $A_{2m+1} = I$, where I is the identity matrix, and $b_{2m+1} = 1$. The objective value in the dual problem, $b^T y$ is then given by the last entry in y.

To attempt to fathom the node, we need to construct a matrix Z from (3.5) using a y of our choice. If the resulting matrix Z is PSD, then the node in the tree is fathomed. The y we chose should correspond to an objective value less than the lower bound given by *best*. Thus we choose

$$y_{2m+1} < best - |I|, \tag{3.8}$$

where we subtract the size of the independent set constructed so far, $|I|$, from the best known solution to give a lower bound on the size of the maximum independent set on the subgraph for this node in the search tree.

To construct the rest of the matrix Z we need to choose the other values in y. Operating under the assumption that the subgraph for this node has not changed drastically from the subgraph of a parent node, we might try to use the y values from a parent node where the BPM or CSDP method was applied to its subgraph. The entries in the previous y correspond to edges in the old subgraph. Because at least one node has been removed from the ancestor's subgraph to form the current subgraph, there are fewer edges in the new subgraph. After the BPM or CSDP is applied, the optimal y is stored as well as the mapping from the subgraph to the original, full graph. Then at any node derived from the current node we can construct a mapping from the new, unbounded subgraph to the old, bounded subgraph and use the stored y values that still correspond to edges in the current subgraph to construct a matrix Z with the desired objective value. If Z is PSD, we can quickly fathom the new, unbounded node.

Rather than storing every y from every node in the branch-and-bound tree that still has unbounded descendants, we chose to simplify the process by storing y from only the most recent bound performed. To make this approach viable, a depth first search must be implemented so that descendant nodes are considered before any others are bounded.

3.4 A warm start for the BPM method

Beyond assuming that the dual y values will be similar to those of an ancestor node, we might also assume that the primal X matrix will be similar as well, see [21] for a detailed discussion of such assumptions. The rows and columns of X correspond to vertices in the subgraph of the current node. Using

the mapping from the last bounded node in the tree to the full graph that is already stored for use of the ancestor y values, rows and columns can be deleted from an ancestor X to yield a new X whose rows and columns correspond to the vertices of the new subgraph. It is well known that deleting corresponding rows and columns from a positive semidefinite matrix yields another positive semidefinite matrix. Thus, initializing Z with the zero matrix means that the KKT conditions

$$\langle X, Z \rangle = 0, \ X \succeq 0, \ Z \succeq 0 \tag{3.9}$$

necessary for the BPM method are still satisfied.

This warm start is not feasible for the interior point method, as the initial solution generated is on the boundary of the positive semidefinite cone of matrices where the interior point method becomes poorly conditioned. The interior point method specifically attempts to push solutions away from this boundary as much as possible.

CHAPTER 4

COMPUTATIONAL RESULTS

All of the code for this project is written in C and compiled using `gcc` version `4.2.4`. `BLAS` and `LAPACK` libraries for `C` are provided by the `ATLAS` package, version `3.8`. Source code for the implementation of BPM for branch and bound is available at `http://euler.nmt.edu/~brian/mis_bpm`. Source code for CSDP, developed by Brian Borchers, is available through `https://projects.coin-or.org/Csdp`. All benchmarks were run on a Dell 490 Workstation with two dual core Intel Xeon 5150 processors and 8 GB of RAM. The machine's operating system is Ubuntu linux with kernel version 2.6.24.

The graphs used for benchmarks are from three sources. The first set is composed of the graphs for the clique benchmarks in the Second DIMACS Challenge [16], available through

`ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/`.

We inverted these graphs for use as benchmarks for the maximum independent set problem. The second set comes from N.J.A. Sloane's challenge problems [29]. These graphs arise from coding theory and provide a sample of large graphs for the maximum independent set problem. The third set was developed

by Ke Xu at the Beijing University of Aeronautics and Astronautics. It consists of random graphs designed to have "hidden" solutions and is available through

`http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm`.

The so called "Benchmarks with Hidden Optimum Solutions for Graph Problems," or BHOSLIB, is unique compared to the DIMACS and Sloane graphs in that the Lovász ϑ-number is the same as the size of the maximum independent set for each.

4.1 Calculating the Lovász Theta Number

Our first comparison is between CSDP and our implementation of BPM outside of branch and bound, calculating ϑ for all the graphs from the DIMACS benchmark problems, Sloane's challenge problems, and the BHOSLIB. Because the interior point method's (and hence, CSDP's) most computationally intensive operation is the solution of a system of m equations, requiring $O(m^3)$ time, we expect that the time to compute the theta number of a graph with CSDP will be dependent on the number of edges in the graph, m. Similarly, the boundary point method requires $O(n^3)$ time for the eigenvalue decomposition, so we expect the BPM to be dependent on the number of vertices in the graph, n. However, we note that these individual operations are performed at each iteration of the BPM and IPM, so the times may vary considerably depending on how many iterations are required for the algorithm to converge for a given graph.

For each graph, Table 4.1 lists the number of vertices and edges, as well as the number of seconds required for BPM and CSDP to converge such that the 2-norm of the normalized residuals of the primal and dual equality constraints are less than 10^{-5}. The final column gives the number of edges divided by the number of vertices in the graph, E/V. Time and memory limits were placed on the calculations. Graphs requiring more than two hours show "$\geq$ 14400" seconds and graphs requiring more than six gigabytes of memory show "*". Times in **bold** correspond to the algorithm that has a clear advantage for the given graph. Bold times are only given when both algorithms were able to determine ϑ. Any time that one algorithm is unable to find ϑ due to memory constraints, the other algorithm holds a clear advantage on this hardware, even if it was not able to find ϑ in the given time.

Table 4.1: CSDP vs. BPM; Calculating the Lovász theta number

	Vertices	Edges	CSDP	BPM	E/V
Sloane challenge graphs					
1dc.64	50	409	**0**	7	8.2
1dc.128	112	1281	**2**	13	11.4
1dc.256	238	3583	**31**	165	15.1
1dc.512	492	9395	469	**341**	19.1
1dc.1024	1002	23645	7294	**1897**	23.6
1dc.2048	2024	57853	*	$\geq$ 14400	28.6
1et.64	62	264	0	0	4.3
1et.128	126	672	**1**	2	5.3
1et.256	254	1664	**4**	37	6.6
1et.512	510	4032	**53**	217	7.9
1et.1024	1022	9600	**548**	3341	9.4
1et.2048	2046	22528	**7465**	$\geq$ 14400	11.0
1tc.128	60	210	**0**	1	3.5
1tc.256	178	952	**1**	18	5.3
1tc.512	424	2858	**18**	152	6.7
1tc.1024	926	7484	**279**	2228	8.1
1tc.2048	1940	18446	**4123**	$\geq$ 14400	9.5
1zc.128	60	210	1	1	3.5
1zc.256	238	2632	10	**6**	11.1
1zc.512	492	6678	133	**80**	13.6
1zc.1024	1002	16350	2258	**859**	16.3
1zc.2048	2024	39072	*	$\geq$ 14400	19.3
1zc.4096	4070	91740	*	$\geq$ 14400	22.5
2dc.128	70	1949	6	8	27.8
2dc.256	182	10883	651	**82**	59.8
2dc.512	420	43413	*	199	103.4
2dc.1024	912	149778	*	1414	164.2
2dc.2048	1914	473507	*	$\geq$ 14400	247.4
Dimacs benchmark graphs					
brock200_1	200	5066	50	**3**	25.3
brock200_2	200	10024	363	**3**	50.1
brock200_3	200	7852	177	**3**	39.3
brock200_4	200	6811	137	**3**	34.1
continued on the next page					

continued from the previous page

	Vertices	Edges	CSDP	BPM	E/V
brock400_1	400	20077	3220	**17**	50.2
brock400_2	400	20014	3199	**17**	50.0
brock400_3	400	20119	3251	**17**	50.3
brock400_4	400	20035	2804	**17**	50.1
brock800_1	800	112095	*	92	140.1
brock800_2	800	111434	*	92	139.3
brock800_3	800	112267	*	93	140.3
brock800_4	800	111957	*	92	139.9
c-fat200-1	200	18366	2172	**5**	91.8
c-fat200-2	200	16665	1696	**28**	83.3
c-fat200-5	200	11427	460	**9**	57.1
c-fat500-1	500	120291	*	88	240.6
c-fat500-2	500	115611	*	87	231.2
c-fat500-5	500	101559	*	265	203.1
c-fat500-10	500	78123	*	352	156.2
hamming6-2	64	192	0	0	3.0
hamming6-4	64	1312	1	0	20.5
hamming8-2	256	1024	**1**	36	4.0
hamming8-4	256	11776	609	**4**	46.0
hamming10-2	1024	5120	**86**	7808	5.0
hamming10-4	1024	89600	*	453	87.5
johnson8-2-4	28	168	0	0	6.0
johnson8-4-4	70	560	0	0	8.0
johnson16-2-4	120	1680	2	0	14.0
johnson32-2-4	496	14880	1010	**15**	30.0
keller4	171	5100	59	**3**	29.8
keller5	776	74710	*	120	96.3
keller6	3361	1026582	*	$\geq$ 14400	305.4
MANN_a9	45	72	0	0	1.6
MANN_a27	378	702	**3**	58	1.9
MANN_a45	1035	1980	**49**	1724	1.9
MANN_a81	3321	6480	**1625**	$\geq$ 14400	2.0
p_hat300-1	300	33917	*	18	113.1
p_hat300-2	300	22922	6014	**87**	76.4
p_hat300-3	300	11460	619	**35**	38.2
p_hat500-1	500	93181	*	49	186.4

continued on the next page

continued from the previous page					
	Vertices	Edges	CSDP	BPM	E/V
p_hat500-2	500	61804	*	325	123.6
p_hat500-3	500	30950	*	138	61.9
p_hat700-1	700	183651	*	128	262.4
p_hat700-2	700	122922	*	824	175.6
p_hat700-3	700	61640	*	486	88.1
p_hat1000-1	1000	377247	*	365	377.2
p_hat1000-2	1000	254701	*	2661	254.7
p_hat1000-3	1000	127754	*	1288	127.8
p_hat1500-1	1500	839327	*	1729	559.6
p_hat1500-2	1500	555290	*	12433	370.2
p_hat1500-3	1500	277006	*	9893	184.7
san200_0.7_1	200	5970	103	**31**	29.9
san200_0.7_2	200	5970	**151**	726	29.9
san200_0.9_1	200	1990	**5**	30	10.0
san200_0.9_2	200	1990	**6**	30	10.0
san200_0.9_3	200	1990	**7**	943	10.0
san400_0.5_1	400	39900	*	318	99.8
san400_0.7_1	400	23940	6850	**340**	59.9
san400_0.7_2	400	23940	6853	**304**	59.9
san400_0.7_3	400	23940	8195	**32**	59.9
san400_0.9_1	400	7980	267	345	20.0
san1000	1000	249000	*	921	249.0
sanr200_0.7	200	6032	83	**3**	30.2
sanr200_0.9	200	2037	5	4	10.2
sanr400_0.5	400	39816	*	15	99.5
sanr400_0.7	400	23931	5477	**16**	59.8
BHOSLIB benchmark graphs					
frb30-15-1	450	17827	2266	**49**	39.6
frb30-15-2	450	17874	2586	**49**	39.7
frb30-15-3	450	17809	2552	**49**	39.6
frb30-15-4	450	17831	2557	**48**	39.6
frb30-15-5	450	17794	2537	**48**	39.5
frb35-17-1	595	27856	*	117	46.8
frb35-17-2	595	27847	*	116	46.8
frb35-17-3	595	27931	*	120	46.9
frb35-17-4	595	27842	*	115	46.8

continued on the next page

continued from the previous page					
	Vertices	Edges	CSDP	BPM	E/V
frb35-17-5	595	28143	*	119	47.3
frb40-19-1	760	41314	*	260	54.4
frb40-19-2	760	41263	*	266	54.3
frb40-19-3	760	41095	*	263	54.1
frb40-19-4	760	41605	*	263	54.7
frb40-19-5	760	41619	*	269	54.8
frb45-21-1	945	59186	*	560	62.6
frb45-21-2	945	58624	*	553	62.0
frb45-21-3	945	58245	*	551	61.6
frb45-21-4	945	58549	*	552	62.0
frb45-21-5	945	58579	*	553	62.0
frb50-23-1	1150	80072	*	1208	69.6
frb50-23-2	1150	80851	*	1176	70.3
frb50-23-3	1150	81068	*	1216	70.5
frb50-23-4	1150	80258	*	1129	69.8
frb50-23-5	1150	80035	*	1135	69.6
frb53-24-1	1272	94227	*	1823	74.1
frb53-24-2	1272	94289	*	1736	74.1
frb53-24-3	1272	94127	*	1727	74.0
frb53-24-4	1272	94308	*	1788	74.1
frb53-24-5	1272	94226	*	1768	74.1
frb56-25-1	1400	109676	*	2521	78.3
frb56-25-2	1400	109401	*	2471	78.1
frb56-25-3	1400	109379	*	2496	78.1
frb56-25-4	1400	110038	*	2515	78.6
frb56-25-5	1400	109601	*	2682	78.3
frb59-26-1	1534	126555	*	4096	82.5
frb59-26-2	1534	126163	*	4242	82.2
frb59-26-3	1534	126082	*	4002	82.2
frb59-26-4	1534	127011	*	4043	82.8
frb59-26-5	1534	125982	*	4240	82.1

Times listed in seconds,

** denotes a graph that required more than 6 GB of memory*

From the table, we can see that our implementation of the BPM does not require more than 6GB of RAM for any of the graphs, while CSDP exceeds this limit for all of the graphs with more than 30,000 edges (almost a third of the graphs). This is one of the most important features of the BPM method; it often requires considerably less memory than interior point methods for the SDP form of the Lovász ϑ problem. With current hardware limitations, the BPM allows us to consider larger graphs.

Considering only graphs to which both algorithms could be applied, the BPM has noticeably smaller running times for most. This is not true for all of the graphs, though. CSDP performs considerably better than the BPM on the 1et, 1tc, MANN, and san200_0.9 graphs. These are graphs where the ratio of edges to vertices is relatively low compared to the other graphs. In fact, when there was a noticeable difference in times, CSDP converged faster for all but two of the graphs where both algorithms were successfully applied and where E/V was less than or equal to 10 (these values of E/V are in **bold**). The two exceptions were cases where E/V was still relatively small.

The charts also lend credence to the hypothesis that CSDP's performance is dependent on the number of edges in the graph and the BPM's on the number of vertices in the graph. This is further evidenced by the plots in Figures 4.1 and 4.2. First, we can note that there appears to be little correlation between times for CSDP and the number of vertices in the graph and no correlation between times for the BPM and the number of edges in the graph. Some correlation between the time for BPM and the number of edges can be seen on a subset of the graphs. This is because the number of edges in

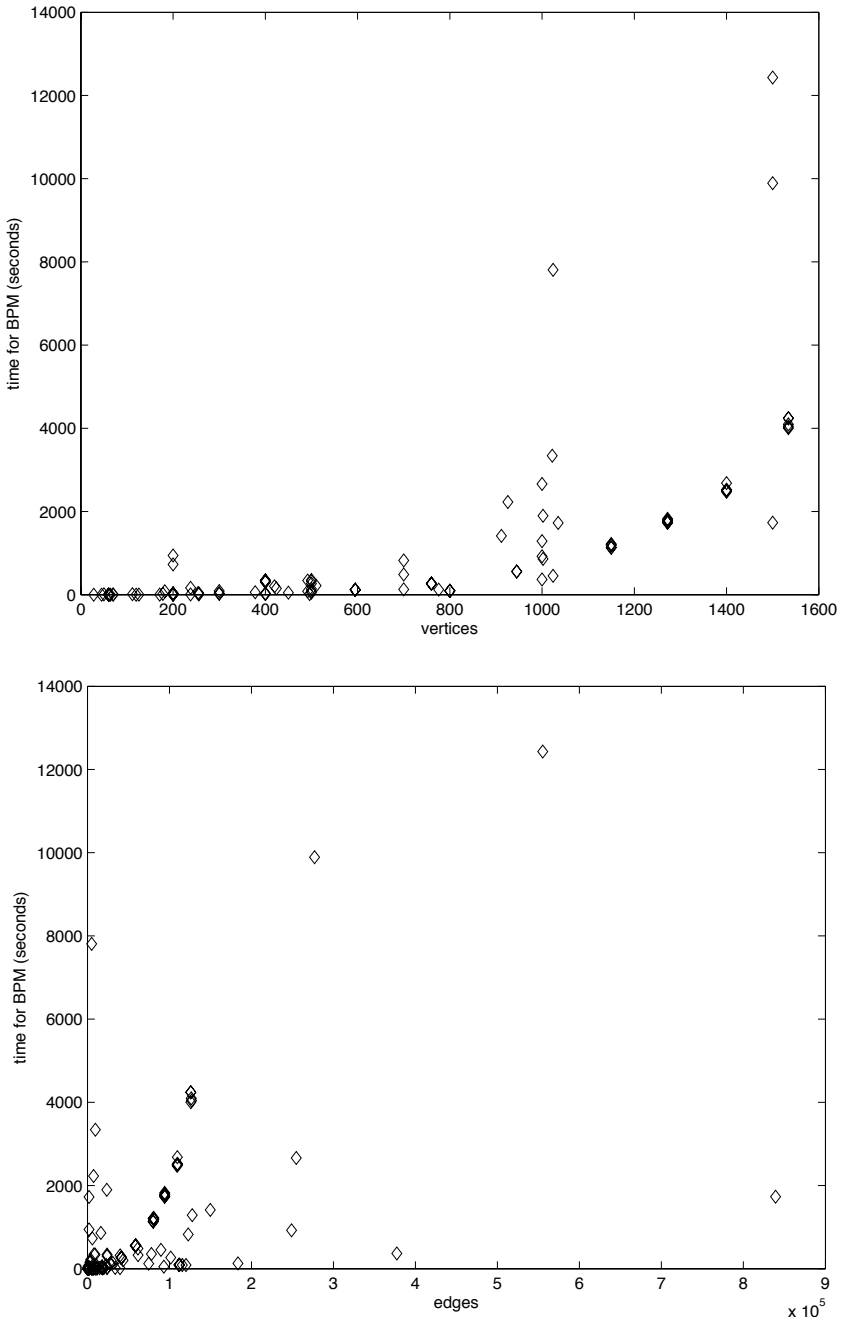

Figure 4.1: Plots showing the time for BPM to compute ϑ versus the number of vertices and versus the number of edges.

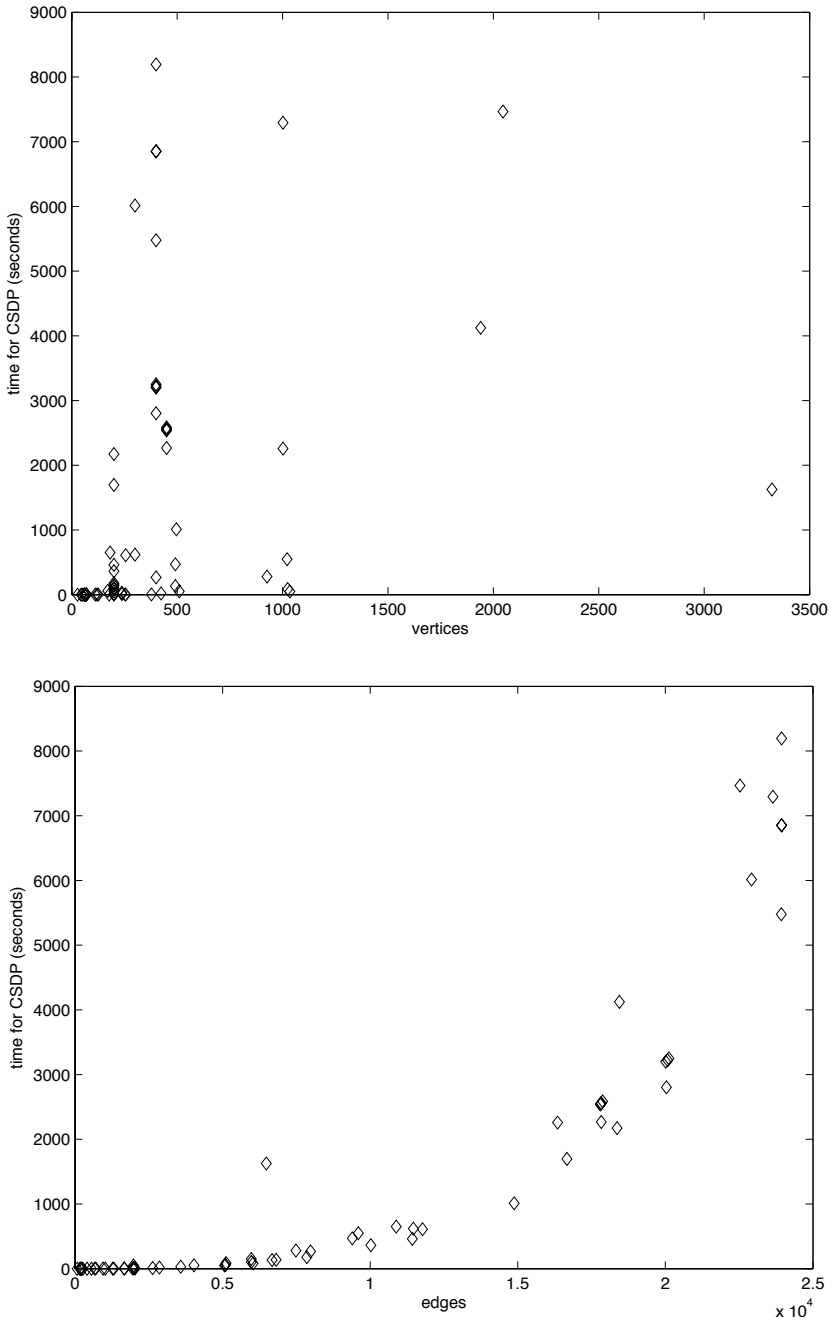

Figure 4.2: Plots showing the time for CSDP to compute ϑ versus the number of vertices and versus the number of edges.

the BHOSLIB problems is directly related to the number of vertices. On the other hand, we can see that the times for CSDP increase with the number of edges and that the times for the BPM increase with the number of vertices, overall. As expected, because only the time for individual iterations of these algorithms are bounded by cubic functions of m and n, exceptions to these dependencies exist. Some graphs require more iterations to achieve convergence than others. Note that the BPM, especially, has varying performance in graphs with the same number of vertices, although the most difficult graphs still seem to require more time with increasing vertices.

To determine if these relationships are indeed polynomial, Figure 4.3 shows the plots of the BPM time versus vertices and CSDP time versus edges on log-log plots. We can see from these that the time for CSDP has a very nearly cubic relationship with the number of edges in the graph. The time for the BPM does not clearly show a polynomial dependence on the number of vertices in the graph, however we can still see some relationship. The differences seen here might be explained by the variance in the number of iterations required for each algorithm to converge. Whereas CSDP required at least five iterations to converge and no more than 13 iterations for all graphs considered here, the BPM required at least 40 iterations and at most 72,000 iterations to converge. Even though individual iterations are bounded by cubic functions of the number of vertices, we might expect to see a large variation in the time for convergence of the BPM due to the high variability in the number of iterations required.

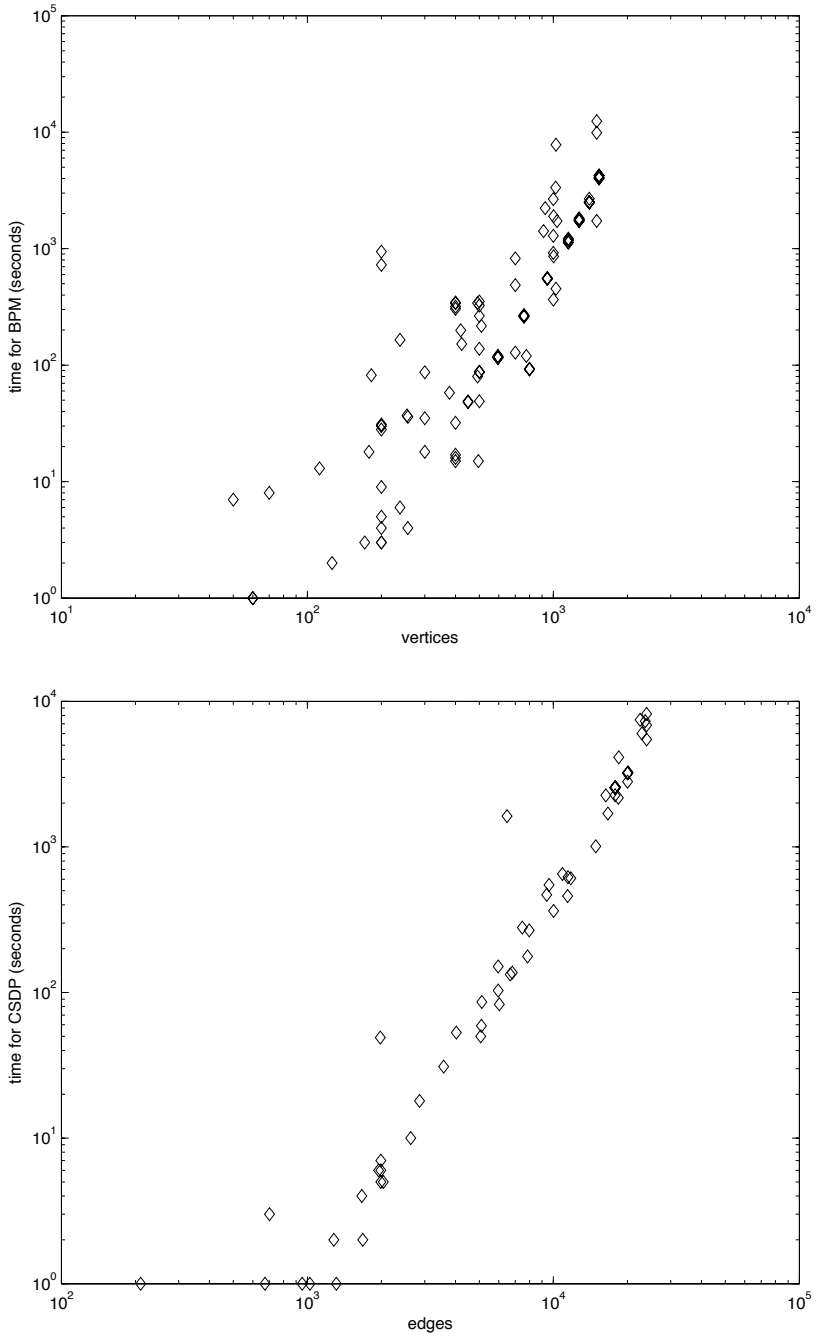

Figure 4.3: Log-log plots showing the time for BPM to compute ϑ versus the number of vertices and for CSDP to compute ϑ versus the number of edges.

4.2 Traversing the branch-and-bound tree

Our second comparison is between CSDP and the BPM (both with and without the use of a warm start) *inside* of the branch-and-bound algorithm. Table 4.2 shows times for each version of the branch-and-bound algorithm to completely traverse the branch-and-bound tree for the given graph. Times listed in **bold** are those for which CSDP or the BPM showed a significant benefit. Only graphs from the DIMACS benchmarks and the BHOSLIB are considered here. For each graph, the initial lower bound on the size of the maximum independent set is listed. The size of a maximum independent set is known for each of these graphs, but this is not necessarily the initial bound provided. A smaller bound might be provided in order to increase the running time of branch and bound, or a larger bound might be provided in order to decrease the running time. Bounds are chosen to keep running times reasonable and force the algorithm to consider numerous nodes in the tree so that we can observe the effects of using a warm start for the BPM.

The final column of Table 4.2 is the time to traverse the tree using the BPM with a warm start divided by the time to traverse the tree without a warm start. Values less than or equal to 0.9, where the warm start significantly reduced the time to traverse the tree, are listed in **bold** and values greater than or equal to 1.1, where the warm start was detrimental to the running time, are listed in *italics*.

Table 4.2: CSDP vs. BPM; Traversing the branch-and-bound tree

	bound	CSDP	BPM	warm BPM	$\frac{\text{warm BPM}}{BPM}$
Sloane challenge graphs					
brock200_1	21	1432	**183**	173	0.95
brock200_2	0	2936	**19**	14	**0.74**
brock200_3	0	1673	**53**	56	1.06
brock200_4	0	1466	**75**	69	0.92
brock400_1	29	≥ 7200	≥ 7200	≥ 7200	
brock400_2	31	≥ 7200	**2117**	2084	0.98
brock400_3	31	≥ 7200	**1913**	1925	1.01
brock400_4	33	≥ 7200	**774**	743	0.96
brock800_1	40	*	1129	1498	*1.33*
brock800_2	40	*	1663	2176	*1.31*
brock800_3	40	*	1452	1868	*1.29*
brock800_4	40	*	1605	2044	*1.27*
c-fat200-1	0	2145	**5**	5	1.00
c-fat200-2	0	1617	**18**	18	1.00
c-fat200-5	0	1151	**13**	13	1.00
c-fat500-1	12	*	91	91	1.00
c-fat500-2	24	*	88	88	1.00
c-fat500-5	62	*	203	202	1.00
c-fat500-10	124	*	195	195	1.00
hamming6-4	0	2	1	1	1.00
hamming8-2	0	**1**	31	31	1.00
hamming8-4	0	598	**4**	4	1.00
hamming10-2	511	**83**	4709	3074	**0.65**
hamming10-4	50	*	2636	1364	**0.52**
johnson8-2-4	0	0	0	0	
johnson8-4-4	0	0	0	0	
johnson16-2-4	0	2	0	0	
johnson32-2-4	0	995	**15**	15	1.00
keller4	0	455	**25**	24	0.96
keller5	30	*	5157	3694	**0.72**
keller6	62	*	≥ 7200	≥ 7200	
MANN_a9	0	0	0	0	
MANN_a27	126	**153**	964	980	1.02

continued on the next page

continued from the previous page					
	bound	CSDP	BPM	warm BPM	$\frac{\text{warm BPM}}{BPM}$
MANN_a45	355	**119**	2636	2657	1.01
p_hat300-1	0	*	80	54	**0.68**
p_hat300-2	0	$\geq$ 7200	**134**	134	1.00
p_hat300-3	36	$\geq$ 7200	**563**	460	**0.82**
p_hat500-1	9	*	773	586	**0.76**
p_hat500-2	36	*	960	813	**0.85**
p_hat500-3	56	*	470	403	**0.86**
p_hat700-1	14	*	367	411	*1.12*
p_hat700-2	48	*	1169	987	**0.84**
p_hat700-3	71	*	989	728	**0.74**
p_hat1000-1	16	*	3960	4133	1.04
p_hat1000-2	54	*	4015	3394	**0.85**
p_hat1000-3	83	*	2560	2014	**0.79**
p_hat1500-1	21	*	$\geq$ 7200	$\geq$ 7200	
p_hat1500-2	76	*	$\geq$ 7200	$\geq$ 7200	
p_hat1500-3	114	*	$\geq$ 7200	$\geq$ 7200	
san200_0.7_1	0	102	**19**	19	1.00
san200_0.7_2	0	149	**36**	36	1.00
san200_0.9_1	0	**5**	17	17	1.00
san200_0.9_2	0	**6**	18	18	1.00
san200_0.9_3	0	**8**	27	27	1.00
san400_0.5_1	0	*	170	168	0.99
san400_0.7_1	0	6767	**134**	131	0.98
san400_0.7_2	0	6779	**143**	142	0.99
san400_0.7_3	20	$\geq$ 7200	**704**	643	0.91
san400_0.9_1	99	458	**173**	167	0.97
san1000	13	*	1192	1160	0.97
sanr200_0.7	0	1389	**110**	102	0.93
sanr200_0.9	42	1799	1795	1788	1.00
sanr400_0.5	13	*	708	730	1.03
sanr400_0.7	25	$\geq$ 7200	**2773**	2500	**0.90**
BHOSLIB benchmark graphs					
frb30-15-1	29	$\geq$ 28800	28587	24462	**0.86**
frb30-15-2	29	$\geq$ 28800	4236	3504	**0.83**
frb30-15-3	29	$\geq$ 28800	194	169	**0.87**
frb30-15-4	29	$\geq$ 28800	$\geq$ 28800	$\geq$ 28800	
continued on the next page					

continued from the previous page					
	bound	CSDP	BPM	warm BPM	$\frac{\text{warm BPM}}{BPM}$
frb30-15-5	29	$\geq$ 28800	5499	4556	**0.83**

Times listed in seconds,

** denotes a graph that required more than 6 GB of memory*

4.2.1 The BPM versus CSDP

Comparing the times for the branch-and-bound algorithm in Table 4.2 using CSDP with those using the BPM shows that the graphs where a significant benefit is seen with one bounding method over another are *exactly* the same as those seen in Table 4.1. This seems to imply that the choice of using BPM or CSDP has no large effect on the branch-and-bound tree, only on the time required for individual bounds within the algorithm. We can investigate this hypothesis by examining the number of nodes in the search tree that are processed within the branch-and-bound algorithm. Table 4.3 shows this for all graphs that branch-and-bound was able to search the entire tree within the memory and time constraints using both BPM and CSDP.

Table 4.3: CSDP vs. BPM; Number of nodes processed in the branch-and-bound search tree

	CSDP	BPM
brock200_1	1863	1733
brock200_2	267	223
brock200_3	533	601
brock200_4	749	745
c-fat200-1	3	3
c-fat200-2	3	3*
c-fat200-5	15	11
hamming6-2	3	3
hamming6-4	23	23
hamming8-2	3	3*
hamming8-4	3	3
hamming10-2	3	63*
johnson8-2-4	3	3
johnson8-4-4	3	3
johnson16-2-4	3	3
johnson32-2-4	3	3
keller4	191	203
MANN_a9	11	11
MANN_a27	4871	13457
MANN_a45	3	3
san200_0.7_1	9	9*
san200_0.7_2	27	29*
san200_0.9_1	13	11*
san200_0.9_2	15	17*
san200_0.9_3	17	21*
san400_0.7_1	11	15*
san400_0.7_2	41	55*
san400_0.9_1	23	25*
sanr200_0.7	1173	1183
sanr200_0.9	7397	9913

*A * denotes a case where the initial calculation of ϑ was truncated before achieving the desired convergence when using the BPM*

For most of the graphs, the number of nodes processed is nearly the same using BPM and CSDP. Notable exceptions are for hamming10-2 and MANN_a27.

In hamming10-2, the difference can be attributed to a flaw in the BPM method, as implemented here. A limit on the number of iterations used for any single bound is enforced within the branch-and-bound algorithm for both CSDP and the BPM . This limit is meant to avoid spending too much time on any single bound. However, when this limit is reached while bounding the root node in the search tree, problems may arise. In §1.3.1 we discussed not calculating ϑ as a bound if a rough estimate of the bound implied that there was no way to fathom the current node by calculating ϑ. The estimate, however, depends on a relatively precise value of ϑ for some parent node in the search tree. For this reason, ϑ is always calculated to full tolerance for the root node. If the maximum iterations is reached, a poor estimate of ϑ may be given and several levels of the search tree may be unnecessarily skipped, greatly increasing the size of the search tree. This is a simple problem to fix, perhaps the constraint on iterations could be ignored for the root node, but it exemplifies the branch-and-bound algorithm's high sensitivity to such decisions.

4.2.2 The BPM with and without a warm start

The leftmost column of Table 4.2 shows the fraction of time used for the branch-and-bound algorithm when a warm start, as discussed in §3.4, is implemented. Because each of these benchmarks were run only once, some variation in the running times should be expected (the computer used for the benchmarks was not solely dedicated to these tests). Arbitrarily, a significance

of a 10% increase or decrease has been noted with *italicized* and **bold** values, respectively. Out of a total of 59 graphs for which the search tree was fully traversed in the alloted time, 36 had no significant change, 18 had decreased times with the warm start, and 5 had increased times. In the 18 that had reduced times, in general the warm start reduced the number of iterations (and time) required for each calculation of ϑ, rarely changing the search tree itself. In the 5 with increased times, 4 were from the brock800 graphs. For these graphs, the warm start increased the number of iterations (and time) needed for many of the calculations of ϑ, but did not significantly change the search tree. In p_hat700-1, use of the warm start actually did reduce the number of iterations needed to calculate ϑ, but a difference in choice of which vertex to branch on resulted in needing to traverse two levels deeper into the branch-and-bound search tree.

4.2.3 The BPM with and without attempting to fathom using parent y-values

For all of the graphs listed in Table 4.2, the branch-and-bound algorithm was applied using an additional modification of the BPM. As discussed in §3.3, the y-values from parent nodes in the search tree were used in an attempt to fathom the current node without using the BPM. In all cases, the Z constructed from the dual equality constraints (with an objective value that would fathom the current node) was not positive semidefinite, hence no nodes were fathomed using this approach.

CHAPTER 5

CONCLUSIONS

In applying the branch-and-bound algorithm to the maximum independent set problem, there are many choices for how to bound the size of the maximum independent set from above. In this thesis, we have considered only the SDP form of the Lovász ϑ number. We compared the benefits and detriments of solving this SDP using the boundary point method, developed by Povh et al. [25], with those when using the more common interior point method.

For a general SDP, both methods may run into similar memory constraints as the size of the graph increases, as both require storage for at least one m by m matrix (where m is the number of edges in the graph). But, when applied to the SDP form ϑ, the m by m matrix becomes diagonal, and the memory requirements for the boundary point method are greatly reduced. The largest matrix required is then an n by n matrix, where n is the number of vertices in the graph and $m \gg n$ for most graphs. Thus the boundary point method allows us to consider graphs larger than an interior point method can when restrictions from current computing hardware are considered.

On graphs where both methods could be applied, neither method was shown to always calculate ϑ faster than the other. Data presented in §4.1 showed that the time for the interior point method to converge is dependent

on m (an individual iteration requires $O(m^3)$ time), while the time for the boundary point method to converge is dependent on n (an individual iteration requires $O(n^3)$ time). Based on the results given here for the implementations of the BPM and IPM used, a rule of thumb was presented based on the fraction m/n. When $m/n > 10$, the boundary point method appeared to converge faster, and when $m/n < 10$, CSDP (based on the interior point method) appeared to converge faster.

Within the branch-and-bound algorithm, the choice of the method to solve the SDP did not seem to have a large effect on the search tree. In most cases, the number of nodes considered in the tree was independent of this choice. Thus, the method that computed a single ϑ value faster for the given graph tended to traverse the search tree faster.

In an attempt to improve the boundary point method within the branch-and-bound algorithm, two modifications were made based on the primal (X) and dual (y,Z) variables obtained at parent nodes in the search tree. First, using y-values from a parent node, we constructed a Z matrix that satisfied the dual equality constraints and had a dual-objective value that would fathom the current node in the search tree. If this Z could be shown to be positive semidefinite, then the (y, Z) pair would be dual feasible and we could fathom the current node without computing ϑ. For the benchmark graphs considered here, all of the attempts to fathom a node based on this method failed. We can conclude that this modification rarely, if ever, reduces running time for the branch-and-bound algorithm. Second, based on the work of Mitchell in [21], we attempted to use X-values from a parent node as a warm start for the BPM.

This modification had more positive results. In most cases, using the warm start resulted in little change to the running time for the branch-and-bound algorithm, but in many it showed a substantial decrease in the time required to compute individual ϑ values. We must note also that, for some graphs, this modification resulted in an increased time for calculating ϑ.

Overall, it appears that we can make several general conclusions based on the results presented here. First, the BPM allows us to calculate ϑ for much larger graphs than does the IPM. Second, for graphs that both methods can calculate ϑ, we have constructed a rule of thumb that allows us to decide which method will require less computational time. And third, it seems that the performance of BPM was improved within the branch-and-bound algorithm through the use of a warm start.

5.1 Further Work

We considered only a small set of benchmark graphs for the results presented here. In order to confirm some of the conclusions made above, a wider variety of graphs should be considered. We would want to test the hypothesis that the value m/n can be used to predict whether the BPM or the IPM will calculate ϑ faster as well as the hypothesis that the warm start improves running times for the BPM. Additionally, we might attempt to examine specific properties of the graphs for which the warm start increases running times in an attempt to decide when the warm start will be beneficial.

The largest benefit to the BPM with respect to the MIS problem is probably its ability to calculate ϑ for much larger graphs than the IPM. This

should allow us to confirm known independent sets for some large graphs as being maximum (or find larger independent sets). The Sloane challenge graphs presented in [29] are an excellent example of some large graphs for which the size of a maximum independent set has yet to be confirmed. With the BPM presented here, we are able to begin traversing the branch-and-bound search tree for these graphs. All that is required is the time to completely traverse the tree. Unfortunately, for many of these graphs, ϑ does not provide a very tight bound and the search trees can become quite large. On the computer that was used for benchmarks in this paper, we have estimated that traversing the tree for one of the smaller Sloane graphs with an unconfirmed MIS size might take weeks. This leads us to consider methods of parallelizing the process.

Unfortunately, the boundary point method does not parallelize very well. Its most computationally demanding operation is an eigenvector decomposition, which consists largely of matrix-vector operations. While these operations can be parallelized, memory bandwidth quickly becomes a limiting factor on current hardware. A more effective way to parallelize will almost certainly be to parallelize the branch-and-bound tree itself. The branch-and-bound algorithm parallelized over distributed memory systems very well by distributing individual calculations of the bounds to individual processors. Parallel branch-and-bound algorithms have provided solutions to some notoriously difficult combinatorial optimization problems in the past decade and development of these algorithms has been extensive, see [30, 32]. Combining the BPM's ability to consider much larger graphs than the IPM with these parallel algorithms should allow us to confirm the size of a maximum independent set for many of the remaining Sloane graphs.

Bibliography

[1] Farid Alizadeh, Jean-Pierre A. Haeberly, and Michael L. Overton. Primal-dual interior-point methods for semidefinite programming: convergence rates, stability and numerical results. *SIAM J. Optimization*, 8(3):746–768, August 1998.

[2] L. Babel. A fast algorithm for the maximum weight clique problem. *Computing*, 52(1):31–38, March 1994.

[3] Dimitri P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Computer Science and Applied Mathematics. Academic Press, Inc., New York, NY, 1982.

[4] Immanuel M. Bomze, Marco Budnich, Panos M. Pardalos, and Marcello Pelillo. The maximum clique problem. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization: Supplemental Volume A*, pages 1–74. Kluwer Academic Publications, 1999.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Combridge University Press, Cambridge, United Kingdom, March 2004.

[6] Etienne de Klerk. *Aspects of Semidefinite Programming: Interior Point Algorithms and Selected Applications*, volume 65 of *Applied Optimization*. Kluwer Academic Publications, Dordecht, The Netherlands, 2002.

[7] Igor Dukanovic and Franz Rendl. Semidefinite programming relaxations for graph coloring and maximal clique problems. *Mathematical Programming*, 109(2-3):345–365, March 2007.

[8] Torsren Fahle. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In *Algorithms — ESA 2002*, volume 2461/2002 of *Lecture Notes in Computer Science*. Springer Berlin/Heidelberg, 2002.

[9] Torsten Fahle. Cost based filtering vs. upper bounds for maximum clique. In *CP-AI-OR '02 Workshop*. Le Croisic/France, 2002.

[10] Mituhiro Fukuda, Masakazu Kojima, and Masayuki Shida. Lagrangian dual interior-point methods for semidefinite programs. *SIAM J. Optimization*, 12(4):1007–1031, 2002.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman San Francisco, 1979.

[12] Michel X. Goemans. Semidefinite programming in combinatorial optimization. *Mathematical Programming*, 79(1-3):143–161, October 1997.

[13] Gerald Gruber and Franz Rendl. Computational experience with stable set relaxations. *SIAM J. Optimization*, 13(4):1014–1028, 2003.

[14] Nebojša Gvozdenović and Monique Laurent. Semidefinite bounds for the stability number of a graph via sums of squares of polynomials. *Mathematical Programming*, 110(1):145–173, 2007.

[15] Christoph Helmberg, Franz Rendl, Robert J. Vanderbei, and Henry Wolkowicz. An interior-point method for semidefinite programming. *SIAM J. Optimization*, 6:342–361, 1996.

[16] David S. Johnson and Michael A. Trick, editors. *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, 1996.

[17] Igor N. Kozin and Miles J. Deegan. A study of the performance of lapack symmetric matrix diagonalizers on multi-core architectures. Technical report, Science and Technology Facilities Council, Warrington, UK, 2007.

[18] László Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25(1):1–7, January 1979.

[19] László Lovász. *An Algorithmic Theory of Numbers, Graphs, and Convexity*. Society for Industrial Mathematics, 1986.

[20] László Lovász and Alexander Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM J. Optimization*, 1(2):166–190, 1991.

[21] John E. Mitchell. Restarting after branching in the SDP approach to max-cut and similar combinatorial optimization problems. *Journal of Combanitorial Optimization*, 5(2):151–166, June 2001.

[22] Renato D. C. Monteiro. First- and second-order methods for semidefinite programming. *Mathematical Programming*, 97(1-2):209–244, 2003.

[23] Jorge Nocedal and Wright Stephen J. *Numerical Optimization*. Springer Science+Business Media, Inc., New York, NY, 1999.

[24] P. M. Pardalos and J. Xue. The maximum clique problem. *Journal of Global Optimization*, 4(3):301–328, 1994.

[25] J. Povh, F. Rendl, and A. Wiegele. A boundary point method to solve semidefinite programs. *Computing*, 78(3):277–286, November 2006.

[26] Jean-Charles Régin. Using constraint programming to solve the maximum clique problem. In *Principles and Practice of Constraint Programming – CP 2003*, volume 2833/2003 of *Lecture Notes in Computer Science*, pages 634–648. Springer Berlin/Heidelberg, 2003.

[27] Silvia Richter, Malte Helmert, and Charles Gretton. A stochastic local search approach to vertex cover. In *KI 2007: Advances in Artificial Intelligence*, volume 4667/2007 of *Lecture Notes in Computer Science*, pages 412–426. Springer Berlin/Heidelberg, 2007.

[28] Alexander Schrijver. A comparison of the delsarte and lovász bounds. *Information Theory, IEEE Transactions on*, 25(4):425–429, 1979.

[29] N. J. A. Sloane. Challenge problems: Independent sets in graphs. http://www.research.att.com/~njas/doc/graphs.html, July 2005.

[30] El-Ghazali Talbi, editor. *Parallel Combinatorial Optimization*. Wiley-Interscience, Hoboken, New Jersey, 2006.

[31] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, 37(1):95–111, January 2007.

[32] W.J. van Hoeve. Parallel branch-and-bound algorithms using semidefinite programming relaxation for the independent set problem. Technical report, University of Twente, October 2000.

[33] E. Alper Yildirim and Xiaofei Fan-Orzechowski. On extracting maximum stable sets in perfect graphs using Lovász's theta function. *Computational Optimization and Applications*, 33(2-3):229–247, 2006.