Optimization Bounds from Binary Decision Diagrams

David Bergman[*] Andre A. Cire[†] Willem-Jan van Hoeve[‡]
J. N. Hooker[§]

April 2012

Abstract

We explore the idea of obtaining bounds on the optimal value of an optimization problem from a discrete relaxation based on binary decision diagrams (BDDs). We show how to construct a BDD that represents a relaxation of an optimization problem with binary variables, and how to obtain a bound for any separable objective function by solving a shortest (or longest) path problem in the BDD. As a test case we apply the method to the maximum independent set problem on a graph. We find that it can can deliver significantly tighter bounds, in far less computation time, than state-of-the-art integer programming software obtains for an integer programming formulation by solving a continuous relaxation augmented with cutting planes.

1 Introduction

Bounds on the optimal value are often indispensible for the practical solution of discrete optimization problems, as for example in branch-and-bound procedures. Such bounds are frequently obtained by solving a continuous relaxation of the problem, perhaps a linear programming (LP) relaxation of an integer programming model. In this paper, we explore an alternative strategy of obtaining bounds from a *discrete* relaxation, namely a binary decision diagram (BDD).

Binary decision diagrams are compact graphical representations of Boolean functions Ake78,Lee59,Bry86. They were originally introduced for applications in circuit design and formal verification [14, 15] but have since been used for a variety of other purposes. These include sequential pattern mining and genetic programming [16, 17].

[*]Tepper School of Business, Carnegie Mellon University, bergman@andrew.cmu.edu
[†]Tepper School of Business, Carnegie Mellon University, acire@andrew.cmu.edu
[‡]Tepper School of Business, Carnegie Mellon University, vanhoeve@andrew.cmu.edu
[§]Tepper School of Business, Carnegie Mellon University, jh38@andrew.cmu.edu

A BDD can represent the feasible set of a 0-1 optimization problem, because the constraints can be viewed as defining a Boolean function $f(x)$ that is 1 when x is a feasible solution. Unfortunately, a BDD that exactly represents the feasible set can grow exponentially in size. We circumvent this difficulty by creating a *relaxed* BBD of limited size that represents a superset of the feasible set. The relaxation is created by merging nodes of the BDD in such a way that no feasible solutions are excluded. A bound on any additively separable objection function can now be obtained by solving a longest (or shortest) path problem on the relaxed BDD. The idea is readily extended to general discrete (as opposed to 0-1) optimization problems by using *multivalued decision diagrams* (MDDs).

As a test case, we apply the proposed method to the maximum independent set problem on a graph. We find that BDDs can deliver significantly tighter bounds than those obtained by state-of-the-art integer programming software, which solves an LP relaxation augmented by cutting planes. The BDD bounds are also obtained in far less computation time.

The paper is organized as follows. After a brief literature review, we show how BDDs can represent 0-1 optimization problems in general and the maximum weighted independent set problem in particular. We then exhibit an efficient top-down compilation algorithm that generates exact reduced BDDs for the independent set problem, and prove its correctness. We then modify the algorithm to generate a limited-size relaxed BDD, prove its correctness, and show that it has polynomial time complexity. We also discuss heuristics for ordering variables and deciding which nodes to merge while building a relaxed BDD.

At this point we report computational results for random and benchmark instances of the maximum independent set problem. We experiment with various heuristics for ordering variables and merging nodes in the relaxed BDDs and test the quality of the bound provided by the relaxed BDDs versus the size allowed for the BDD. We then compare the bounds obtained from the BDDs with those obtained at the root node by the CPLEX mixed-integer solver for a 0-1 programming formulation of the problem. We conclude with suggestions for future work.

2 Previous Work

Relaxed BDDs and MDDs were introduced by [2] for the purpose of replacing the typical domain store relaxation used in constraint programming by a richer data structure. They found that MDDs drastically reduce the size of the search tree and allow much faster solution of problems with multiple all-different constraints, which are equivalent to graph coloring problems. Similar methods were applied to other types of constraints in [12] and [13]. The latter paper also develops a general top-down compilation method based on state information at nodes of the MDD.

None of this work addresses the issue of obtaining bounds from relaxed

BDDs. Three of us applied this idea to the set covering problem in a conference paper [6], which reports good results for certain structured instances. In the current paper, we present novel and improved methods for BDD compilation and relaxation. These methods are superior to continuous relaxation technology for a much wider range of instances, and require far less time.

The ordering of variables can have a significant bearing on the effectiveness of a BDD relaxation. We investigate this for the independent set problem in [5] and apply the results here.

Binary decision diagrams have also been applied to post-optimality analysis in discrete optimization [10, 11], cut generation in integer programming [3], and 0-1 vertex and facet enumeration [4].

3 Binary Decision Diagrams

Given binary variables $x = (x_1, \ldots, x_n)$, a *binary decision diagram* (BDD) $B = (U, A)$ for x is a directed acyclic multigraph that encodes a set of values of x. The set U of nodes is partitioned into layers $L_1, \ldots, L_n$ corresponding to variables $x_1, \ldots, x_n$, plus a terminal layer L_{n+1}. Layers L_1 and L_{n+1} are singletons consisting of the *root* node r and the *terminal* node t, respectively. All directed arcs in A run from a node in some layer L_j to a node in some deeper layer L_k ($j < k$). For a node $u \in L_j$, we write $\ell(u) = j$ to indicate the layer in which u lies.

Each node $u \in L_j$ has one or two out-directed arcs, a *0-arc* $a_0(u)$ and/or a *1-arc* $a_1(u)$. These correspond to setting x_j to 0 and 1, respectively. We use the notation $b_0(u)$ to indicate the node at the opposite end of arc $a_1(u)$, and similarly for $b_1(u)$. Each arc-specified path from r to t represents the 0-1 tuple x in which $x_{\ell(u)} = 1$ for each 1-arc $a_1(u)$ on the path, and $x_j = 0$ for all other j. The entire BDD represents the set $\mathrm{Sol}(B)$ of all tuples corresponding to r–t paths.

Figure 1(a) illustrates a BDD for variables $x = (x_1, \ldots, x_6)$. The left-most path from root node r to terminal node t represents the tuple $(x_1, \ldots, x_6) = (0, 0, 1, 0, 0, 0)$. The third arc in the path is a *long arc* because it skips one or more variables. It represents the partial assignment $(x_3, x_4, x_5, x_6) = (1, 0, 0, 0)$.[1] In general, a long arc from level j to level k encodes the partial assignment $(x_j, \ldots, x_{k-1}) = (1, 0, \ldots, 0)$. The entire BDD of Fig. 1(a) represents a set of 10 tuples, corresponding to the 10 r–t paths.

Given nodes $u, u' \in U$, we will say that $B_{uu'}$ is the subgraph of B induced by the nodes in U that lie on some directed path from u to u'. Thus $B_{rt} = B$. Two nodes u, u' on a given level of a BDD are *equivalent* if $B_{ut} = B_{u't}$. A *reduced* BDD is one that contains no equivalent nodes. A standard result of BDD theory is that for a fixed variable order, there is a unique reduced BDD

[1]This differs from the standard interpretation of a long arc (u, u'), which represents all partial assignments to $(x_{\ell(u)}, \ldots, x_{\ell(u')})$ with $x_{\ell(u)} = 1$ if (u, u') is a 1-arc, and all partial assignments with $x_{\ell(u)} = 0$ if it is a 0-arc.

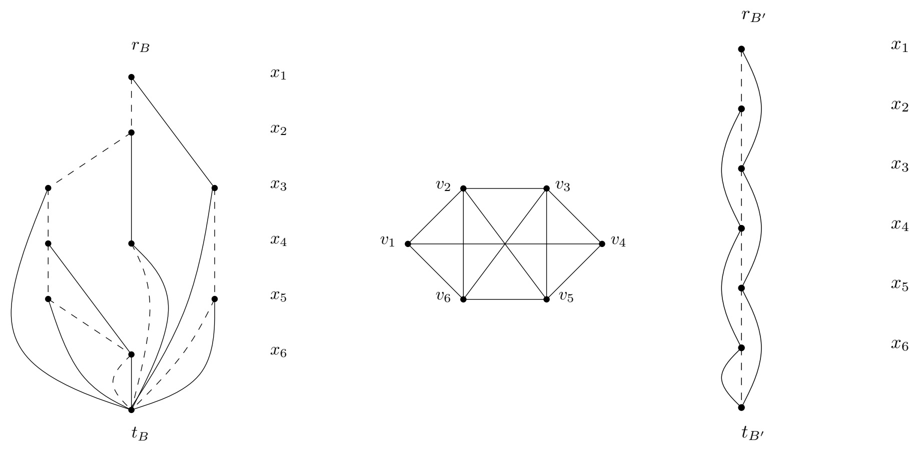


Figure 1: (a) Example of a BDD. (b) Independent set problem for which (a) is an exact BDD. (c) Relaxed BDD for the problem in (b).

that represents a given set. The *width* ω_j of level j is $|L_j|$, and the width $\omega(B)$ of a BDD B is $\max_j \omega_j$. The BDD of Fig. 1(a) is reduced and has width 2.

The feasible set of any optimization problem with binary variables $x_1, \ldots, x_n$ can be represented by an appropriate reduced BDD. The BDD can be regarded as a compact representation of a search tree for the problem. It can in principle be obtained by omitting infeasible leaf nodes from the search tree, superimposing isomorphic subtrees, and identifying all feasible leaf nodes with t. We will present below a much more efficient procedure for obtaining a reduced BDD. A slight generalization of BDDs, *multivalued decision diagrams* (MDDs), can similarly represent the feasible set of any discrete optimization problem. MDDs allow a node to have more than two outgoing arcs and therefore accommodate discrete variables with several possible values.

4 BDD Representation of Independent Sets

We focus on BDD representations of the *maximum weighted independent set problem.* Given a graph $G = (V, E)$, an *independent set* is a subset of the vertex set V, such that no two vertices are connected by an edge in E. If each vertex v_i is associated with a weight w_i, the problem is to find an independent set of maximum weight. If each $w_i = 1$, we have the *maximum independent set problem.*

If we let binary variable x_j be 1 when v_j is included in the independent set, the feasible solutions of any independent set problem can be represented by a BDD on variables $x_1, \ldots, x_n$. Figure 1(a), for example, represents the 10 independent sets of the graph in Fig. 1(b).

We can remove any node u in a BDD with a single outgoing arc if it is a 0-arc

$a_0(u)$. This is accomplished by replacing every 0-arc $a_0(u')$ for which $b_0(u') = u$ with a longer arc $a_0(u')$ for which $b_0(u') = b_0(u)$, and similarly for every such 1-arc. If the BDD represents an independent set problem, a single outgoing arc must be a 0-arc, which means that all nodes with single outgoing arcs can be removed. Every node in the resulting BDD has exactly two outgoing arcs.

To represent the objective function in the BDD, let each 1-arc $a_1(u)$ have length equal to the weight $w_{\ell(u)}$, and each 0-arc length 0. Then the length of a path from r to t is the weight of the independent set it represents. The weighted independent set problem becomes the problem of finding a longest path in a BDD. If all the weights $w_i = 1$, the four longest paths in the BDD of Fig. 1(a) have length 2, corresponding to the maximum independent sets $\{v_1, v_3\}$, $\{v_1, v_5\}$, $\{v_2, v_4\}$, and $\{v_4, v_6\}$.

Any binary optimization problem with an additively separable objective function $\sum_j f_j(x_j)$ can be similarly represented as a longest path problem on a BDD. Long edges as defined here may be used if $f_j(0) = 0$ and $f_j(1) \geq 0$ for each j. This condition is met by any independent set problem with nonnegative weights. It can be met by any binary problem if each $f_j(x_j)$ is replaced with $\bar{f}(\bar{x}_j)$, where $\bar{f}_j(0) = 0$ and

$$\begin{array}{lll} \bar{f}_j(1) = f_j(1) - f_j(0) \ \text{ and } \ \bar{x}_j = x_j, & & \text{if } f_j(1) \geq f_j(0) \\ \bar{f}_j(1) = f_j(0) - f_j(1) \ \text{ and } \ \bar{x}_j = 1 - x_j, & & \text{otherwise} \end{array}$$

5 Exact and Relaxed BDDs

If Sol(B) is equal to the feasible set of an optimization problem, we will say that B is an *exact BDD* for the problem. If Sol(B) is a superset of the feasible set, B is a *relaxed BDD* for the problem. We will construct *limited-width* relaxed BDDs by requiring $\omega(B)$ to be at most some pre-set maximum width W.

Figure 1(c) shows a relaxed BDD B' of width 1 for the independent set problem of Fig. 1(b). B' represents 21 vertex sets, including the 10 independent sets. The length of a longest path in B' is therefore an upper bound on the optimal value of the original problem. If, again, all weights $w_j = 1$, the longest path length is 3, which provides an upper bound on the maximum cardinality 2 of an indepedent set.

6 Exact BDD Compilation

We now describe an algorithm that builds an exact reduced BDD for an independent set problem. Similar algorithms can be designed for any optimization problem on binary variables [13], but here we exploit the special structure of the independent set problem.

Starting with the root r, the procedure constructs the BDD $B = (U, A)$ layer by layer, selecting a graph vertex for each layer and associating a state with each node. The definition of a state is problem dependent, but for the independent set problem we define it as follows. Using a slight abuse of notation, let Sol(B)

be the set of independent sets represented by B (rather than the corresponding set of tuples x). Thus, in particular, $\mathrm{Sol}(B_{ru})$ is the set of independent sets defined by paths from r to u. Let the *neighborhood* $N(\bar{V})$ of a vertex set $\bar{V}$ be the set of vertices adjacent to vertices in $\bar{V}$, where by convention $\bar{V} \subset N(\bar{V})$. The *state* $s(u)$ of node u is the set of vertices that can be added to any of the independent sets defined by paths from r to u. Thus

$$s(u) = \{v_{\ell(u)}, \ldots, v_n\} \setminus \bigcup_{\bar{V} \in \mathrm{Sol}(B_{ru})} N(\bar{V})$$

In an exact BDD, all paths to a given node u define partial assigments to x that have the same feasible completions. So $s(u) = \{v_{\ell(u)}, \ldots, v_n\} \setminus N(\bar{V})$ for any $\bar{V} \in \mathrm{Sol}(B_{ru})$. In addition, no two nodes on the same layer of an exact reduced BDD have the same feasible completions. So we have the following:

Lemma 1 *An exact BDD for G is reduced if and only if $s(u) \neq s(u')$ for any two nodes u, u' on the same layer of the BDD.*

The exact BDD compilation is stated in Algorithm 1. We begin by creating the root r of B, which has state $s(r) = V$ because every vertex in V is part of some independent set. We then add r to a pool P of nodes that have not yet been placed on some layer. Each node $u \in P$ is stored along with its state $s(u)$ and the arcs that terminate at u.

To create layer L_j, we first select the j-th vertex v_j by means of a function `select` (step 4), which can follow a predefined order or select vertices dynamically. We let L_j contain the nodes $u \in P$ for which $v_j \in s(u)$. These are the only nodes in P that will have both outgoing arcs $a_0(u)$ and $a_1(u)$. All of the remaining nodes in P would have only an outgoing 0-arc if placed on this layer and can therefore be skipped. The nodes in L_j are removed from P, as we need only process them once.

For each node u in L_j, we create outgoing arcs $a_0(u)$ and $a_1(u)$ as follows. Node $b_0(u)$ (i.e., the node at the opposite end of $a_0(u)$) has state $s_0 = s(u) \setminus \{v_j\}$, and node $b_1(u)$ has state $s_1 = s(u) \setminus N(\{v_j\})$. To ensure that the BDD is reduced, we check whether $s_0 = s(u')$ for some node $u' \in P$, and if so let $b_0(u) = u'$. Otherwise, we create node u_0 with $s(u_0) = s_0$, let $b_0(u) = u_0$, and insert u_0 into P. If $s_0 = \emptyset$, u_0 is the terminal node t. Arc $a_1(u)$ is treated similarly. After the last iteration, P will contain exactly one node with state $\emptyset$, and it becomes the terminal node t of B.

We now show this algorithm returns the exact BDD.

Theorem 2 *For any graph $G = (V, E)$, Algorithm 1 generates a reduced exact BDD for the independent set problem on G.*

Proof. Let $\mathrm{Ind}(G)$ be the set of independent sets of G. We wish to show that if B is the BDD created by Algorithm 1, $\mathrm{Sol}(B) = \mathrm{Ind}(G)$. We proceed by induction on $n = |V|$.

Algorithm 1 Exact BDD Compilation

```
1: Create node $r$ with $s(r) = V$
2: Let $P = \{r\}$ and $R = V$
3: for $j = 1$ to $n$ do
4:    $v_j = \text{select}(R, P)$
5:    $R \leftarrow R \backslash \{v_j\}$
6:    $L_j = \{u \in P : v_j \in s(u)\}$
7:    $P \leftarrow P \backslash L_j$
8:    for all $u \in L_j$ do
9:       $s_0 := s(u)\backslash\{v_j\}$, $s_1 := s(u)\backslash N(v_j)$
10:      if $\exists\, u' \in P$ with $s(u') = s_0$ then
11:         $a_0(u) = (u, u')$
12:      else
13:         create node $u_0$ with $s(u_0) = s_0$ ($u_0 = t$ if $s_0 = \emptyset$)
14:         $a_0(u) = (u, u_0)$
15:         $P \leftarrow P \cup \{u_0\}$
16:      if $\exists\, u' \in P$ with $s(u') = s_1$ then
17:         $a_1(u) = (u, u')$
18:      else
19:         create node $u_1$ with $s(u_1) = s_1$ ($u_1 = t$ if $s_0 = \emptyset$)
20:         $a_1(u) = (u, u_1)$
21:         $P \leftarrow P \cup \{u_1\}$
22: Let $t$ be the remaining node in $P$ and set $L_{n+1} = \{t\}$
```

First, suppose $n = 1$, and let G consist of a single vertex v. B consists of two nodes, r and t, and two arcs $a_0(r)$ and $a_1(r)$, both directed from r to t. Therefore, $\text{Sol}(B) = \{\emptyset, v\} = \text{Ind}(G)$. Moreover, this BDD is trivially reduced.

For the induction hypothesis, suppose that Algorithm 1 creates a reduced exact BDD for any graph on fewer than n (≥ 2) vertices. Let G be a graph on n vertices. Assume the `select` function in Step 4 returns vertices in a fixed order $v_1, \ldots, v_n$. Let $G_0 = (V_0, E_0)$ be the subgraph of G induced by vertex set $V\backslash\{v_1\}$, and $G_1 = (V_1, E_1)$ the subgraph induced by $V\backslash N(v_1)$. Then $\text{Ind}(G) = \text{Ind}(G_0) \cup \{\bar{V} \cup \{v_1\} \mid \bar{V} \in \text{Ind}(G_1)\}$, since each independent set either excludes v_1 (whereupon it appears in $\text{Ind}(G_0)$) or includes v_1 (whereupon it appears as the union of $\{v_1\}$ with a set in $\text{Ind}(G_1)$).

Let B be the BDD returned by the algorithm for G. By construction, $s(b_0(r)) = V_0$ and $s(b_1(r)) = V_1$. Let B_0 be the BDD that the algorithm creates for G_0, and similarly for B_1. We observe as follows that $B_0 = B_{b_0(r)t}$ and $B_1 = B_{b_1(r)t}$. The root r_0 of B_0 has $s(r_0) = V_0$, the same state as node $b_0(r)$ in B. But the successor nodes created by the algorithm for r_0 and $b_0(r)$ depend entirely on the state and are therefore identical in B_0 and B, respectively. Moreover, the states of the successor nodes depend entirely on the state of the parent and which branch is taken. Thus the successor nodes have the same states in B_0 as in B. If we apply this reasoning recursively, we obtain

$B_0 = B_{b_0(r)t}$. A parallel argument shows that $B_1 = B_{b_1(r)t}$. Now

$$\begin{aligned} \text{Sol}(B) &= \text{Sol}(B_{b_0(r)t}) \cup \{\bar{V} \cup \{v_1\} \mid \bar{V} \in \text{Sol}(B_{b_1(r)t})\} \\ &= \text{Sol}(B_0) \cup \{\bar{V} \cup \{v_1\} \mid \bar{V} \in \text{Sol}(B_1)\} \\ &= \text{Ind}(G_0) \cup \{\bar{V} \cup \{v_1\} \mid \bar{V} \in \text{Ind}(G_1)\} \\ &= \text{Ind}(G) \end{aligned}$$

as claimed, where the third equation is due to the inductive hypothesis. Furthermore, since all nodes with the same state are merged, Lemma 1 implies that B is reduced. □.

Lemma 3 *The time complexity of Algorithm 1 is polynomial in the size of the reduced exact BDD $B = (U, A)$ for a graph G.*

Proof. Assume for sake of clarity that `select` (Step 4) takes constant time. We observe that an arc of B is never rechecked again once it was created in one of the Steps 11, 14, 17, or 20. Hence, the complexity of the algorithm is dominated by the operations required when creating the out-arcs of a node removed from the pool P.

These operations consist of creating a new state (Step 9) and inserting or searching in the node pool (Steps 10, 15, 16, and 21), which can be implemented in $O(|V|)$. Since every node has exactly two outgoing arcs (i.e., $|A| = 2|U|$), the worst-case complexity of Algorithm 1 is $O(|U|\,|V|)$. □

7 Relaxed BDDs

Limited-width relaxed BDDs allow us to represent an over-approximation of the family of independent sets of a graph, and thus obtain an upper bound on the optimal value of the independent set problem.

We propose a novel top-down compilation method for constructing relaxed BDDs. The procedure modifies Algorithm 1 by forcing nodes to be merged when a particular layer exceeds a pre-set maximum width W. This modification is given in Algorithm 2, which is to be inserted immediately after line 7 in Algorithm 1.

The procedure is as follows. We begin by checking if $\omega_j > W$, which indicates that the width of layer L_j exceeds W. If so, we select a subset M of L_j using function `select_nodes` in Step 2, which ensures that $2 \leq |M| \leq \omega_j - W$. The set M represents the nodes to be merged so that the desired width is met. Various heuristics for selecting M are discussed in Section 8.

The state of the new node that results from the merge, s_{new}, must be such that no feasible independent set is lost in further iterations of the algorithm. As it will be shown in Theorem 4, it suffices to let s_{new} be the union of the states associated with the nodes in M (Step 3). Once s_{new} is created, we search for some node $u' \in L_j$ such that $s(u') = s_{new}$. If u' exists, then by Lemma 1 we are

Algorithm 2 Node merger for obtaining a relaxed BDD.
Insert immediately after line 7 of Algorithm 1.

```
1:  while $\omega_j > W$ do
2:      $M :=$ node_select$(L_j)$  // where $2 \leq |M| \leq \omega_j - W$
3:      $s_{\text{new}} := \bigcup_{u \in M} s(u)$
4:      $L_j \leftarrow L_j \backslash M$
5:      if $\exists u' \in L_j$ with $s(u') = s_{\text{new}}$ then
6:          merge$(M, u')$
7:      else
8:          Create node $\hat{u}$ with $s(\hat{u}) = s_{\text{new}}$
9:          merge$(M, \hat{u})$
10:         $L_j = L_j \cup \{\hat{u}\}$
```

Algorithm 3 merge(M, u')

```
1:  for all $u \in M$ do
2:      for all arcs $a_0(w)$ with $b_0(w) = u$ do
3:          $b_0(w) \leftarrow u'$
4:      for all arcs $a_1(w)$ with $b_1(w) = u$ do
5:          $b_1(w) \leftarrow u'$
```

only required to direct the incoming arcs of the nodes in M to u', as presented in Algorithm 3. Otherwise, we create a new node $\hat{u}$ with $s(\hat{u}) = s_{new}$ and add it to L_j.

In each iteration of the *while* loop in Algorithm 2, we decrease the size of L_j by at least $|M| - 1$. Thus, after at most $\omega_j - W$ iterations, the layer L_j will have width no greater than W. The modified Algorithm 1 hence yields a limited-width W BDD, i.e. $\omega(B) \leq W$.

The correctness of Algorithm 2 is proved as follows.

Theorem 4 *For any graph $G = (V, E)$, Algorithm 1 modified by adding Algorithm 2 after line 7 generates a relaxed BDD.*

Proof. We will use the notation B_u for the BDD consisting of all r–t paths in B that pass through u. Thus

$$\text{Sol}(B_u) = \{V_1 \cup V_2 \ \mid V_1 \in \text{Sol}(B_{ru}),\ V_2 \in \text{Sol}(B_{ut})\} \tag{1}$$

It suffices to show that each iteration of the while-loop yields a relaxed BDD if it begins with a relaxed BDD. Thus we show that if B is a relaxed (or exact) BDD, then the BDD $\hat{B}$ that results from merging the nodes in M satisfies $\text{Sol}(B) \subseteq \text{Sol}(\hat{B})$. Here M is any proper subset of L_j for an arbitrary $j \in \{2, \ldots, n-1\}$.

Let $M = \{u_1, \ldots, u_k\}$ be the nodes to be merged into $\hat{u}$. Also, let $\bar{B}$ be the BDD consisting of all r–t paths in B that do *not* include any of the nodes u_i.

Then

$$\mathrm{Sol}(B) = \mathrm{Sol}(\bar{B}) \cup \bigcup_{i=1}^{k} \mathrm{Sol}(B_{u_i})$$

The merge procedure has no effect on $\mathrm{Sol}(\bar{B})$. Hence it remains to show that

$$\bigcup_{i=1}^{k} \mathrm{Sol}(B_{u_i}) \subseteq \mathrm{Sol}(\hat{B}_{\hat{u}})$$

But we can write

$$\begin{aligned}
\bigcup_{i=1}^{k} \mathrm{Sol}(B_{u_i}) &= \bigcup_{i=1}^{k} \{V_1 \cup V_2 \mid V_1 \in \mathrm{Sol}(B_{ru_i}),\ V_2 \in \mathrm{Sol}(B_{u_i t})\} \\
&= \left\{ V_1 \cup V_2 \;\middle|\; V_1 \in \bigcup_{i=1}^{k} \mathrm{Sol}(B_{ru_i}),\ V_2 \in \bigcup_{i=1}^{k} \mathrm{Sol}(B_{u_i t}) \right\} \\
&= \left\{ V_1 \cup V_2 \;\middle|\; V_1 \in \mathrm{Sol}(\hat{B}_{r\hat{u}}),\ V_2 \in \bigcup_{i=1}^{k} \mathrm{Sol}(B_{u_i t}) \right\} \\
&\subseteq \left\{ V_1 \cup V_2 \;\middle|\; V_1 \in \mathrm{Sol}(\hat{B}_{r\hat{u}}),\ V_2 \in \mathrm{Sol}(\hat{B}_{\hat{u}t}) \right\} \\
&= \mathrm{Sol}(\hat{B}_{\hat{u}})
\end{aligned}$$

The first and last equations are due to (1). The third equation is due to $\bigcup_i \mathrm{Sol}(B_{ru_i}) = \mathrm{Sol}(\hat{B}_{r\hat{u}})$, which follows from the fact that $\hat{u}$ receives precisely the paths received by the u_is before the merge. The fourth line is due to $\bigcup_i \mathrm{Sol}(B_{u_i t}) \subseteq \mathrm{Sol}(\hat{B}_{\hat{u}t})$. This follows from the facts that (a) $\mathrm{Sol}(B_{u_i t})$ contains the independent sets in the subgraph of G induced by $s(u_i)$; (b) $\mathrm{Sol}(\hat{B}_{\hat{u}t})$ contains the independent sets in the subgraph induced by $s(\hat{u})$; and (c) $s(u_i) \subseteq s(\hat{u})$ for all i. $\square$

The time complexity of Algorithm 2 is highly dependent on the `node_select` function and on the number of nodes to be merged. Once a subset M of nodes has been chosen, taking the union of the states (Step 3) has a time complexity of $O(|M||V|)$, and Algorithm 3 has a worst-case time complexity of $O(W|M|)$ by supposing that every node in M is adjacent to as many as W nodes located in previous layers. Hence, if k is the number of nodes to be merged, the complexity of Algorithm 2 is $O(H(k)+|M||V|+W|M|)$ per iteration of the *while* loop in Step 1, where $H(k)$ is the complexity of the node selection heuristic (`node_select`) for a given k. The number of iterations depends on the size of the selected node set. For example, if $|M|$ is always 2, then at most $W - k$ iterations are required (if none of the newly defined states appeared in L_j previously). The time complexity for the complete relaxation procedure is given by the following lemma.

Lemma 5 *Let S be the time complexity of selecting the next variable (`select` function in Step 4 of Algorithm 1), and let $R(k)$ be the time complexity of Algorithm 2. The worst-case time complexity of Algorithm 1 modified with the procedure in Algorithm 2 is given by $O(n(S + R(nW) + W|V|))$.*

Proof. If k nodes are removed from the pool in Step 6 of Algorithm 1, then the merging procedure in Algorithm 2 ensures that at most $2 \min\{k, W\}$ new nodes are added back to the pool. Thus, at each iteration the pool can be increased by at most W nodes. Since n iterations in the worst case are required for the complete compilation, the pool can have at most nW nodes.

Suppose now nW nodes are removed from the pool (Step 6 of Algorithm 1) at a particular iteration. These nodes are first merged so that the maximum width W is met (Algorithm 2), and then new nodes or arcs are created according to the result of the merge. The time complexity for the first operation is $R(nW)$, which yields a new layer with at most W nodes. For the second operation, we observe as in Lemma 3 that creating a new state or searching in the pool size can be implemented in time $O(|V|)$; hence, the second operation has a worst-case time complexity of $O(W|V|)$.

This implies that the time required per iteration is $O(S + R(nW) + W|V|)$, yielding a time complexity of $O(n(S + R(nW) + W|V|))$ for the modified procedure. □

8 Merging Heuristics

The selection of nodes to merge in a layer that exceeds the maximum alloted width W is critical for the construction of relaxation BDDs. Different selections may yield dramatic differences on the obtained upper bounds on the optimal value, since the merging procedure adds paths corresponding to infeasible solutions to the BDD.

In this section we present a number of possible heuristics for selecting nodes. This refers to how the subsets M are chosen on line 2 in Algorithm 2. The heuristics we test are described below.

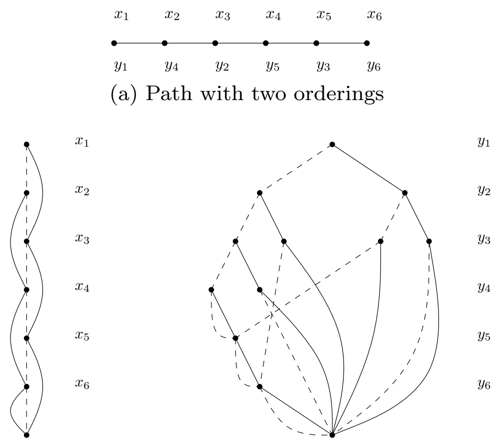
random :	Randomly select a subset M of size $\lvert L_j\rvert - W + 1$ from L_j. We suggest this selection not only as a stand alone selection but also as a heuristic that may be mixed with any of the following heuristics for the purpose of generating several relaxations.
minLP :	Sort nodes in L_j in increasing order of the longest path value up to those nodes and merge the first $\lvert L_j\rvert - W + 1$ nodes. We suggest this selection because infeasibility is introduced into the BDD only when nodes are merged. By selected nodes with the smallest longest path, we are losing information in sections of the BDD where the optimal solution is unlikely to lie.
minSize :	Sort nodes in L_j in decreasing order of their corresponding state sizes and merge the first 2 nodes until $\lvert L_j\rvert \leq W$. With this selection heuristic, we merge nodes that have the largest number of vertices in their associated state. Therefore, it will be likely that they agree on the nodes in their states, and hence this heuristic tends to merge nodes that represent similar regions of the solution space.

9 Variable Ordering

The ordering of the vertices plays an important role in not only the size of exact BDDs, but also in the bound obtained by Relaxation BDDs. It is well known that finding orderings which minimize the size of BDDs (or even improving on a given ordering) is NP-hard [9, 7]. Moreover, in preliminary computational results, we found that the ordering of the vertices is the single most important parameter in creating small width exact BDDs and in proving tight bounds via relaxed BDDs.

Indeed, different orderings can yields exact BDDs with dramatically different widths. For example, Figure 2a shows a path on 6 vertices with two different orderings given by $x_1, \ldots, x_6$ and $y_1, \ldots, y_6$. In Figure 2b we see that the vertex ordering $x_1, \ldots, x_6$ yields an exact BDD with width 1, while in Figure 2c the vertex ordering $y_1, \ldots, y_6$ yields an exact BDD with width 4. This last example can be extended to a path with $2n$ vertices, yielding a BDD with a width of 2^{n-1} while ordering the vertices according to the order that they lie on the paths yields a BDD of width 1.

A thorough analysis of vertex ordering was done by the authors in [5]. We showed that, for particular structured instances of graphs, there exists orderings for which the width of the exact reduced BDD is polynomial in the size of the graph. This includes paths, cliques, and trees. For the case of general graphs, we proved that the ordering induced by a maximal path decomposition of the vertices yields an exact reduced BDD for which the width of layer L_j is bounded by the $j + 1$-st Fibonacci number.


x_1 x_2 x_3 x_4 x_5 x_6
y_1 y_4 y_2 y_5 y_3 y_6
x_1 x_2 x_3 x_4 x_5 x_6
y_1 y_2 y_3 y_4 y_5 y_6

(a) Path with two orderings

(b) x (c) y

Figure 2: Path with two different variable orderings.

The previous paper also discusses various ordering heuristics for relaxed BDDs. We outline them below, noting that the first two orderings are *dynamic*, in that we select the j-th vertex in the order based on the first $j-1$ vertices chosen and the partially constructed BDD. In contrast, the last ordering is *static*, in that the ordering is determined prior to building the BDD.

random : Randomly select some vertex that has yet to be chosen. We suggest this vertex selection not only as a stand alone variable ordering hueristic, but also as a heuristic that may be mixed with any of the following heuristics for the purpose of generating several relaxations.

minState : Select the next vertex v_j as the vertex appearing in the fewest number of states in P. This selection minimizes the size of L_j, given the previous selection of vertices $v_1, \ldots, v_{j-1}$, since the only nodes in P that will appear in L_j are exactly those nodes containing v_j in their associated state. Doing so limits the number of merging operations that need to be performed.

MPD : As mentioned above, it was shown in [5] that a Maximal Path Decomposition of the vertices in a graph yields an ordering which bounds the exact BDD width by the Fibonacci numbers, which grow slower than 2^j (the worst case). Hence this ordering limits the width of all layers, therefore also limiting the number of merging operations necessary to build the BDD.

10 Computational Experiments

In this section we perform an experimental analysis to assess the impact of different parameters on the bounds provided by a relaxation BDD. We first compare the different node merging techniques discussed in Section 8. We then evaluate the bounds obtained for the variable orderings heuristics presented in Section 9. Next, we analyze the strength of the bound with respect to the maximum allotted width. Finally, we compare our bounds with the ones obtained via state-of-the art integer programming technology, which is a common bounding technique for 0-1 problems. We measure the deviation of the bounds by the bound divided by the optimal value (substituting the best lower bound known in the case when the problem is unsolved).

We tested our procedure on two sets of instances. The first set, denoted by `random`, consists of 45 randomly generated graphs according to the Erdös-Rényi model $G(n, p)$, in which graphs on n vertices are constructed in a way that two vertices are adjacent with probability p. We fixed $n = 100$ and generated 5 instances for each p in the set $\{0.1, 0.2, \ldots, 0.9\}$. The second set of instances, denoted by `dimacs`, is composed by the complement graphs of the well-known DIMACS benchmark for the Maximum Clique Problem, obtained from http://cs.hbg.psu.edu/txn131/clique.html. These are graphs of size between 100 and 4000 vertices and contain various types of structure. Furthermore, we consider the misp optimization problem for our test-bed (i.e., $w_j = 1$ for all vertices).

The tests ran on an Intel Xeon E5345 with 8 GB RAM. The BDD was implemented in C++. We used Ilog CPLEX 12.4 as our Integer Programming solver. In particular, we took the bound obtained from the root node relaxation after 1 hour of CPU time. We set the solver parameters in a way to balance the quality of the bound value and the CPU time to process the root node. The CPLEX parameters used in our computational experiments are presented in Table 1.

Table 1: CPLEX Parameters

Parameters (CPLEX internal name)	*Value*
Version	12.4
Presolve (`Preind`)	0 (off)
Number of explored nodes (`NodeLim`)	0 (only root)
Parallel processes (`Threads`)	1
Clique cuts (`Cliques`)	2 (aggressive)
Other cuts (`Covers`, `DisjCuts`, ...)	-1 (off)
Emphasis (`MIPEmphasis`)	2 (optimization)
Primal Heuristics (`HeurFreq`, `RINSHeur`, `Probe`)	-1 (off)
Time limit (`TiLim`)	3600

Two parameters specifically warrant discussion. Our tests set `Preind` to 0 in order to have a fair comparison between the two bounding techniques. Actually, allowing CPLEX to invoke presolving mechanisms typically increases the bound on this problem domain, or returns a bound within 1 of the bound ouputted by turning presolve off. In addition, we turned off all cuts besides clique cuts. This was set because allowing all classes of cuts to be generated often leads to

CPU times orders of magntitude higher and returns a bound typically within 1 of the bound with only clique cuts.

10.1 Merging Heuristics

We tested the three merging heuristics presented in Section 8 on the `random` instance set. In all experiments, we set a maximum width of $W = 10$ and fixed the `MPD` as the ordering of variables for all cases. Figure 3 presents the deviation from the optimum solution for each instance, where the x-axis represents the graph density. In particular, we ran the `random` merging five times per instance, and the results are presented in a candlestick format where the top, center, and bottom represent the maximum, average, and minimum bound obtained, respectively.

We see that among the merging heuristics tested, **minLP** achieves by far the tightest bounds. This behavior follows from the fact that infeasibility is introduced only at those nodes selected to be merged, and it seems better to preserve the nodes with the best bounds, which is accomplished by **minLP**. The plot also highlights the importance of considering structured merging heuristics, since `random` yielded much weaker bounds in comparison to the other techniques tested.

In light of these results, **minLP** is henceforth fixed as the merging heuristic for the remainder of the experiments.

10.2 Variable Ordering Heuristics

We tested the three variable ordering heuristics presented in Section 9 on the `random` instance set. Figure 4 presents the deviation from the optimum solution for each instance, again with the x-axis representing graph density. The maximum alloted width is fixed to $W = 10$. Analogous to the merging heuristic comparison, we ran the `random` ordering five times per instance, and plot the results in a candlestick format.

We see that the `MinState` ordering is the best ordering tested, with this being particularly true for sparse graphs, since the number of possible node states generated by dense graphs is relatively small. As such, we fix this as the ordering for the remainder of the experiments.

10.3 Relation between Bounds and Maximum Allotted Width

The purpose of this experiment is to analyze the impact on the bound provided by the BDDs of the maximum width W imposed on the layers of the BDD. To this end, we tested the bound that Relaxation BDDs provide versus the maximum width W allowed for the relaxations on the `p-hat_300-1` instance from the `dimacs` set. We note that the obtained results were common among all instances tested.

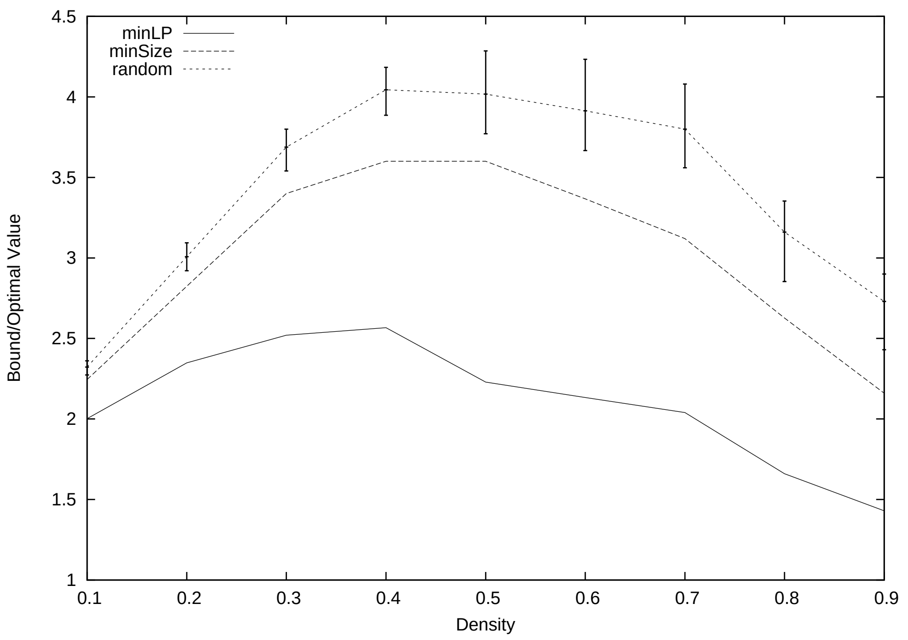


Figure 3: Deviation from the optimal value for each merging heuristic. The results correspond to the `random` instance set with `MPD` ordering and width $W = 10$.

The experiment proceeded as follows. Starting with $W = 5$, we increased W until the bound provided by the Relaxation BDD matched the optimal value (in this case, 8). Figure 5 depicts the result, with the width-axis in log-scale. We see that as the width increases, the bound approaches the optimal value, but that the convergence is super-exponential in W.

10.4 Comparison with CPLEX

The results comparing CPLEX and the BDD approach for the `random` instance set are presented in Table 3. In that table, ρ refers to the graph density, OPT is the average optimal value over all random graphs generated, z_{LP} is the root node relaxation value provided by CPLEX, and W_{10}, W_{50} and W_{100} report the average bound provided by the relaxed BDDs with width 10, 50, and 100, respectively.

Figure 6 depicts plots comparing the deviation from optimum and the CPU time for a maximum alloted width of $W = 100$. We note that the bounds provided by the relaxed BDD are tighter for high density graphs, since the number of possible node states is greatly reduced for such graphs. Moreover, the relaxed BDD time is consistently below 0.08 seconds, with the time required for CPLEX always exceeded this time and in addition deteriorates as the density increases, perhaps due to higher complexity of generating `Clique` cuts in these

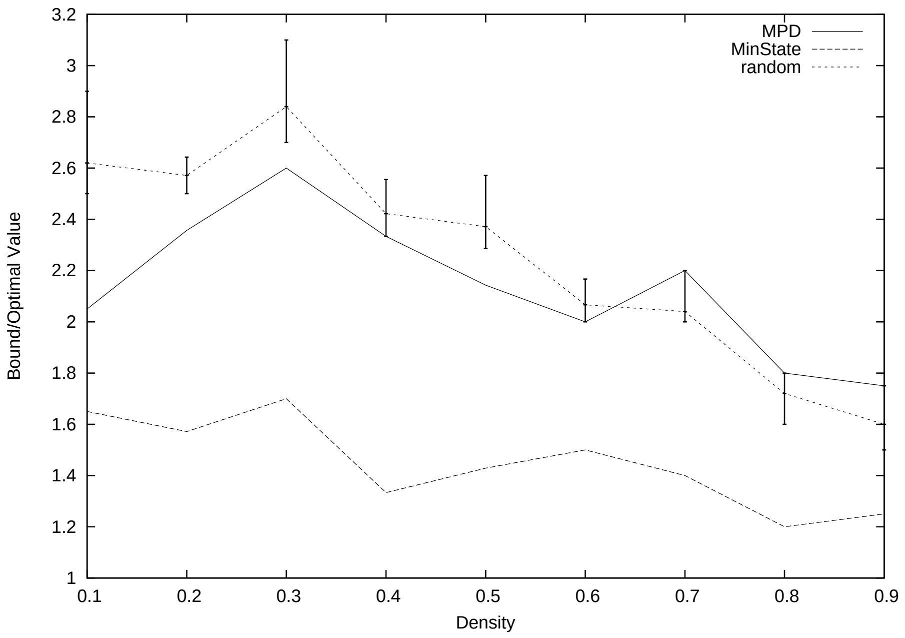


Figure 4: Deviation from the optimal value for each variable ordering heuristic. The results correspond to the `random` instance set with `minLP` merging and width $W = 10$.

cases.

The results for the `dimacs` instance set are presented in Table 4. OPT, ρ and z_{LP} are as in Table 3, where in addition t_{LP} is the time CPLEX took in solving the root node relaxation and n is the number of vertices in the corresponding instance. We report the bound provided by a relaxed BDD having maximum widths $W = 1, 10, 100$, and 1000. Additionally we include t_W, the time required to compile a relaxed BDD with width W.

Figure 7 depicts the plots comparing the deviation from optimum and the CPU time between CPLEX and the BDD approach for the `dimacs` instances and maximum alloted width $W = 1000$. In both plots, the instances are ordered by decreasing performance (measured by deviation) of CPLEX performance. We see that in general the relaxed BDDs provide a significantly better bound than CPLEX. This is especially true for larger instances as seen in Table 4. Also of interest is that typically the times to compute the bounds are quite smaller for the BDD approach. The average time for CPLEX is 564.93 seconds, while for the Relaxation BDDs it is 0.05, 0.33, 2.43, and 22.83 for $W = 1, 10, 100, 1000$, respectively. We note here that for the instances in class `c-fat` we are able to build the exact BDD and that the exact reduced BDDs always have width 4.

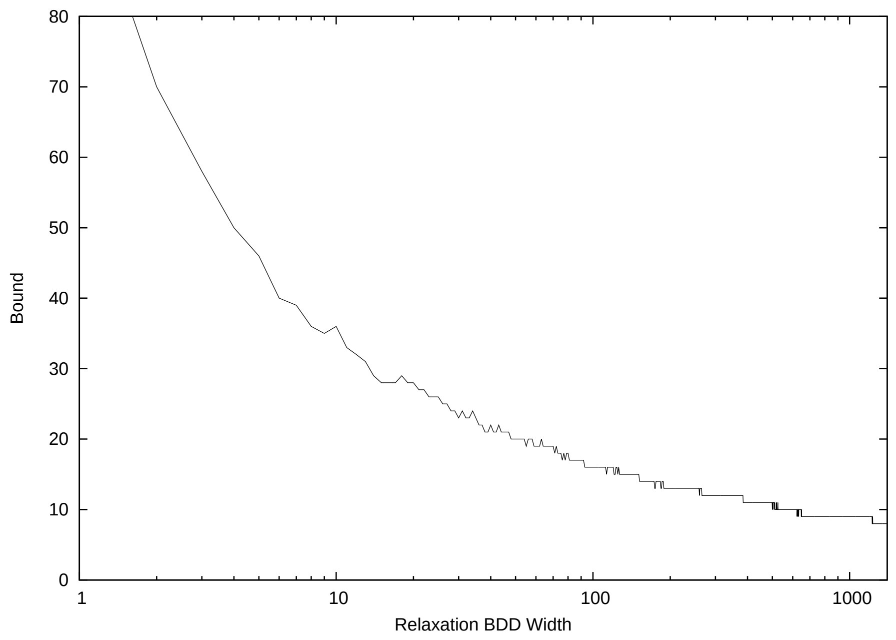


Figure 5: Relaxation Bound versus W for `p-hat_300-1`.

Table 2: Bound comparison between CPLEX and BDD approach on the `random` instance set.

ρ	OPT	CPLEX	W_{10}	W_{50}	W_{100}
0.1	20.6	27.6086	32.4	29	27
0.2	14	19.372	23	18.8	17.6
0.3	10	14.9824	16.8	13.6	12.6
0.4	8.2	12.1355	13	10	9.4
0.5	7	10.1535	10.4	8.4	7.6
0.6	6	8.5885	8.4	6	6
0.7	5	7.41869	6.8	5	5
0.8	5.2	6.49913	6	5.2	5.2
0.9	4.4	5.69734	5	4.4	4.4

11 Conclusions

[ROUGH DRAFT] We proposed a method, based on binary decision diagrams (BDDs), for obtaining relaxation bounds on the optimal value of 0-1 optimization problems. We applied the technique to the maximum independent set problem and found that, in most instances, it provides significantly tighter bounds than state-of-the-art integer programming software obtains by solving an LP relaxation (with cutting planes) at the root node. The BDD bound requires far less computation time, even though we accelerated the integer programming

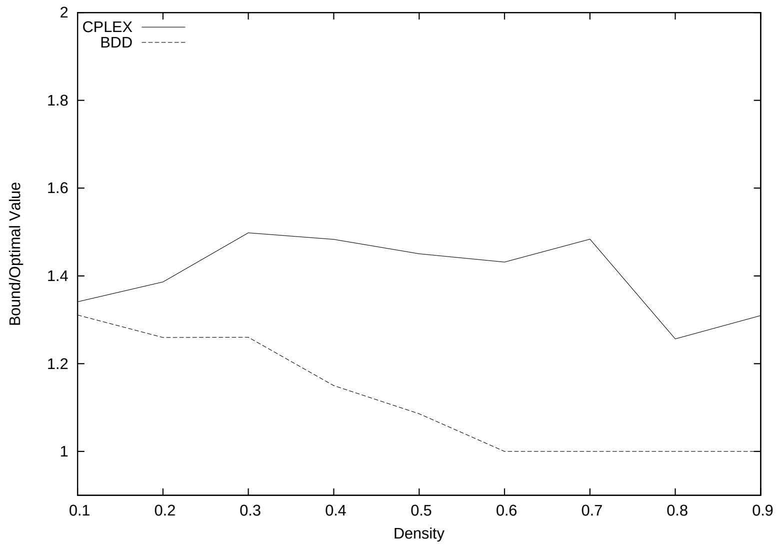
Table 3: Comparison on Random Graphs

			W=10					W=100				
				Rand		MPD	MIN		Rand		MPD	MIN
ρ	OPT	z_{LP}	min	avg	max			min	avg	max		
0.1	20.6	27.6086	50	52.76	55.8	41.2	32.4	38.4	40.4	43	35	27
0.2	14	19.372	36.2	37.32	38.4	32.8	23	23.2	25.04	26.6	23.6	17.6
0.3	10	14.9824	25.4	27.76	29.8	25.2	16.8	15	16.28	17.2	16	12.6
0.4	8.2	12.1355	19.8	21.2	22	21	13	11	11.56	12.4	11.4	9.4
0.5	7	10.1535	15.4	16.32	17.4	15.6	10.4	8	8.44	9	8.4	7.6
0.6	6	8.5885	11.8	12.68	13.6	12.8	8.4	6	6.4	7.2	6.4	6
0.7	5	7.41869	9.2	10.04	11	10.2	6.8	5	5.04	5.2	5	5
0.8	5.2	6.49913	7.6	8.2	8.6	8.6	6	5.2	5.2	5.2	5.2	5.2
0.9	4.4	5.69734	6	6.6	7.2	6.2	5	4.4	4.4	4.4	4.4	4.4

software by turning off the generation of cuts we found to be ineffective.

These results suggest that BDD-based relaxations may have promise as a general technique for bounding the optimal value of discrete problems. Due to the small computation times, they could be used alongside the LP relaxation, at the root node and perhaps at subsequent nodes of a branch-and-bound tree, to obtain tighter bounds with little additional time investment. Alternatively, they could be applied to combinatorial problems that are not formulated as mixed integer models. Unlike LP relaxations, BDD relaxations do not presuppose that the constraints have linear inequality form.

Future research would involve testing of BDD-based methods on additional 0-1 problems, and extending them to general discrete problems by means of multivalued decision diagrams (MDDs). Important subtasks include the development of general variable-ordering and node-merging heuristics. There may be value in specializing BDD-based relaxations to problems in inequality form, for inclusion in existing mixed integer solvers. One can also construct restricted MDDs, as opposed to relaxed MDDs, and use them as a basis for primal heuristics. A more distant goal would be to extend MDD-based technology to accommodate continuous as well as discrete variables.



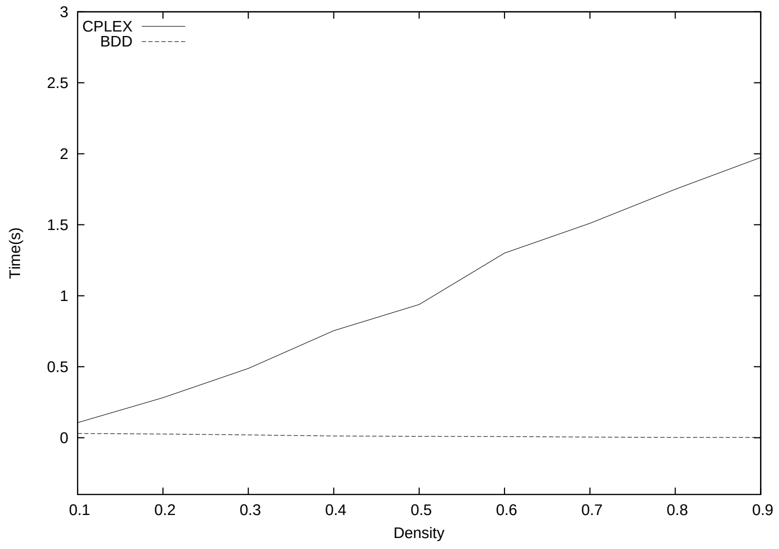
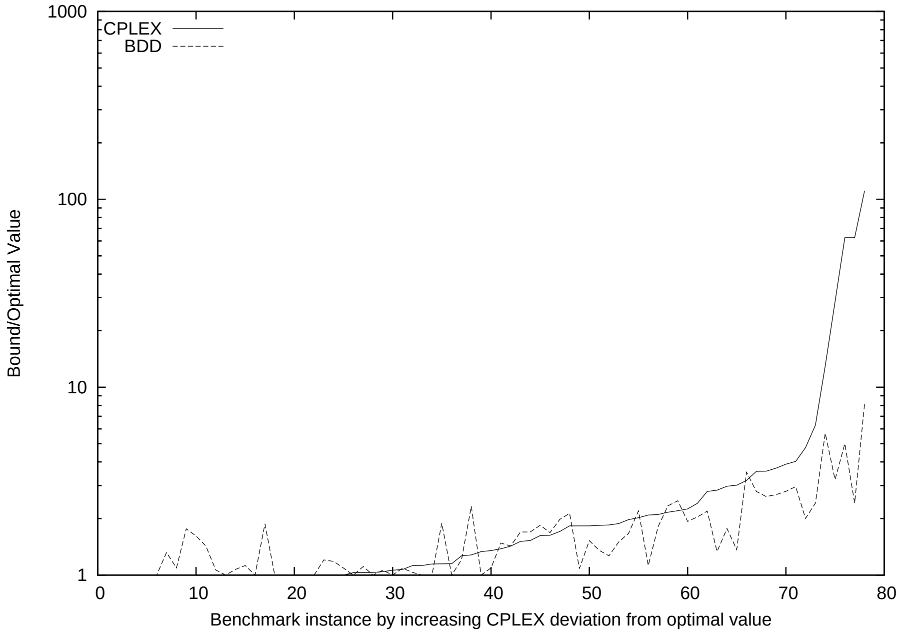


Figure 6: Comparison of deviation from optimum (top) and time (bottom) between CPLEX and BDD approach for the `random` set. The maximum width was set to $W = 100$.



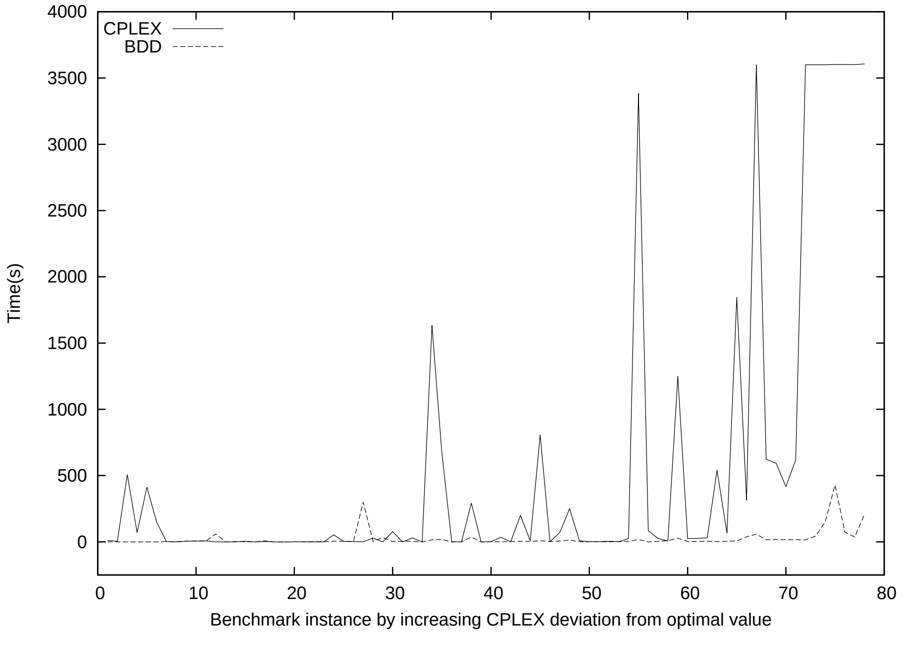


Figure 7: Comparison of deviation from optimum (top) and time (bottom) between CPLEX and BDD approach for the `dimacs` instance set. The maximum width was set to $W = 1000$.

Table 4: Bound comparison for the `dimacs` instance set.

	OPT	n	ρ	z_{LP}	t_{LP}	$W = 1$	t_1	$W = 10$	t_{10}	$W = 100$	t_{100}	$W = 1000$	t_{1000}
brock200_1.clq	21	200	0.25	38.39	1.69	75	0	46	0.02	38	0.11	32	0.91
brock200_2.clq	12	200	0.5	21.93	8.14	50	0	25	0.01	17	0.06	13	0.51
brock200_3.clq	15	200	0.39	27.73	4.26	57	0	34	0.02	25	0.08	19	0.75
brock200_4.clq	17	200	0.34	31.28	1.91	62	0.01	37	0.02	29	0.09	23	0.87
brock400_1.clq	27	400	0.25	65.05	26.31	140	0.01	89	0.07	66	0.42	55	3.83
brock400_2.clq	29	400	0.25	65.21	25.15	134	0.01	89	0.06	66	0.43	56	3.79
brock400_3.clq	31	400	0.25	65.05	26.02	137	0.01	88	0.06	67	0.42	56	3.74
brock400_4.clq	33	400	0.25	65.11	25.02	133	0.01	90	0.07	67	0.42	55	3.99
brock800_1.clq	23	800	0.35	92.56	618.35	216	0.04	126	0.22	89	1.34	68	15.97
brock800_2.clq	24	800	0.35	93.35	416.84	223	0.04	129	0.24	90	1.3	67	16.46
brock800_3.clq	25	800	0.35	92.71	594.02	224	0.04	123	0.23	88	1.3	67	18.11
brock800_4.clq	26	800	0.35	92.85	623.49	225	0.04	128	0.23	89	1.29	68	16.45
C1000.9.clq	68	1000	0.1	217.04	314.08	409	0.07	319	0.54	270	4.34	240	38
C125.9.clq	34	125	0.1	43.06	0.04	62	0	49	0.01	46	0.07	41	0.51
C2000.5.clq	16	2000	0.5	1000.00	3601.88	414	0.24	200	1.17	124	6.1	80	73.77
C2000.9.clq	77	2000	0.1	1000.00	3600.49	800	0.27	618	2.15	507	17.44	438	151.52
C250.9.clq	44	250	0.1	71.67	0.98	121	0	92	0.03	84	0.27	74	2.47
C4000.5.clq	18	4000	0.5	2000.00	3606.94	796	0.92	374	4.76	237	24.53	147	209.03
C500.9.clq	57	500	0.1	123.11	6.47	219	0.02	174	0.13	146	1.07	133	9.13
c-fat200-1.clq	12	200	0.92	12.00	11.57	12	0	12	0	12	0.01	12	0
c-fat200-2.clq	24	200	0.84	24.00	5.07	24	0	24	0	24	0	24	0.01
c-fat200-5.clq	58	200	0.57	66.67	1.46	68	0	58	0.01	58	0.01	58	0
c-fat500-1.clq	14	500	0.96	14.00	507.05	14	0.01	14	0.01	14	0.02	14	0.01
c-fat500-10.clq	126	500	0.63	126.00	69.88	126	0.02	126	0.02	126	0.02	126	0.02
c-fat500-2.clq	26	500	0.93	26.00	413.38	26	0.01	26	0.02	26	0.01	26	0.01
c-fat500-5.clq	64	500	0.81	64.00	147.73	64	0.01	64	0.01	64	0.01	64	0.01
gen200_p0.9_44.clq	44	200	0.1	44.00	0.39	86	0	70	0.02	63	0.17	58	1.43
gen200_p0.9_55.clq	55	200	0.1	55.00	0.28	95	0	76	0.02	64	0.17	60	1.58
gen400_p0.9_55.clq	55	400	0.1	55.00	4.18	130	0.01	126	0.08	102	0.71	97	5.94
gen400_p0.9_65.clq	65	400	0.1	65.00	5.53	155	0.01	128	0.09	113	0.69	105	6.6
gen400_p0.9_75.clq	75	400	0.1	75.00	7.91	169	0.02	135	0.08	114	0.69	107	5.85
hamming10-2.clq	512	1024	0.01	512.00	0.04	555	0.07	567	0.66	536	6.07	545	60.23
hamming10-4.clq	40	1024	0.17	51.20	292.85	173	0.07	154	0.49	126	3.77	93	35.92
hamming6-2.clq	32	64	0.1	32.00	0.01	35	0	34	0	32	0.01	32	0.11
hamming6-4.clq	4	64	0.65	5.33	0.1	11	0	4	0	4	0	4	0
hamming8-2.clq	128	256	0.03	128.00	0.01	138	0	137	0.04	131	0.31	137	3.02
hamming8-4.clq	16	256	0.36	16.00	4.25	45	0	32	0.02	23	0.13	18	1.09
johnson16-2-4.clq	8	120	0.24	8.00	0.06	15	0	15	0.01	13	0.03	8	0.13
johnson32-2-4.clq	16	496	0.12	16.00	2.26	31	0.01	33	0.11	30	0.92	30	7.05
johnson8-2-4.clq	4	28	0.44	4.00	0	7	0	4	0	4	0	4	0
johnson8-4-4.clq	14	70	0.23	14.00	0.02	21	0	17	0	14	0.01	14	0.06
keller4.clq	11	171	0.35	14.85	1.44	28	0	21	0.01	14	0.07	12	0.43
keller5.clq	27	776	0.25	31.00	677.79	71	0.04	74	0.26	62	1.82	51	18.29
keller6.clq	59	3361	0.18	1680.50	3601.81	154	0.72	240	5.5	195	45.47	190	429.03
MANN_a27.clq	126	378	0.01	135.00	0.03	144	0.01	169	0.07	152	0.55	137	4.49
MANN_a45.clq	345	1035	0	360.00	0.08	375	0.04	437	0.38	376	3.27	366	32.42
MANN_a81.clq	1100	3321	0	1134.00	0.36	1161	0.4	1493	3.53	1429	35.82	1222	299.96
MANN_a9.clq	16	45	0.07	18.00	0	21	0	20	0	18	0	16	0.01
p_hat1000-1.clq	10	1000	0.76	47.68	3600.44	134	0.05	55	0.18	32	0.92	20	15.4
p_hat1000-2.clq	46	1000	0.51	92.84	3385.19	247	0.06	154	0.29	122	1.56	101	17.69
p_hat1000-3.clq	68	1000	0.26	149.65	1249.82	350	0.06	245	0.43	197	2.85	169	27.78
p_hat1500-1.clq	12	1500	0.75	750.00	3601.54	207	0.11	80	0.43	48	2.47	29	36.14
p_hat1500-2.clq	65	1500	0.49	408.55	3600.74	379	0.14	237	0.7	187	3.97	157	43.16
p_hat1500-3.clq	94	1500	0.25	335.08	3600.52	507	0.15	357	1	299	6.75	262	58.43
p_hat300-1.clq	8	300	0.76	16.70	82.95	52	0.01	20	0.02	12	0.06	9	0.31
p_hat300-2.clq	25	300	0.51	34.49	34.27	83	0.01	55	0.03	43	0.13	37	1.23
p_hat300-3.clq	36	300	0.26	54.95	9.31	113	0.01	80	0.04	67	0.24	61	2.54
p_hat500-1.clq	9	500	0.75	25.48	539.9	77	0.02	31	0.05	19	0.2	12	2.02
p_hat500-2.clq	36	500	0.5	54.38	198.9	130	0.01	86	0.07	70	0.39	61	4.8
p_hat500-3.clq	50	500	0.25	85.32	69.56	186	0.02	131	0.11	111	0.71	99	6.8
p_hat700-1.clq	11	700	0.75	33.10	1845.98	101	0.03	41	0.09	25	0.4	15	6.18
p_hat700-2.clq	44	700	0.5	71.40	808.32	185	0.03	121	0.14	92	0.77	81	8.58
p_hat700-3.clq	62	700	0.25	113.30	250.18	255	0.04	183	0.21	152	1.39	132	14.5
san1000.clq	15	1000	0.5	17.18	1633.5	27	0.06	23	0.24	17	1.42	15	15.12
san200_0.7_1.clq	30	200	0.3	30.00	0.76	50	0.01	35	0.01	30	0.09	30	0.77
san200_0.7_2.clq	18	200	0.3	18.53	2.11	36	0.01	24	0.01	20	0.08	18	0.55
san200_0.9_1.clq	70	200	0.1	70.00	0.08	108	0	83	0.02	71	0.16	70	1.49
san200_0.9_2.clq	60	200	0.1	60.00	0.14	98	0	77	0.03	65	0.17	60	1.39
san200_0.9_3.clq	44	200	0.1	44.00	0.28	81	0	63	0.02	57	0.17	53	1.67
san400_0.5_1.clq	13	400	0.5	13.80	77.29	24	0.01	15	0.04	13	0.22	13	1.45
san400_0.7_1.clq	40	400	0.3	41.25	28.86	84	0.01	55	0.06	44	0.39	40	3.88
san400_0.7_2.clq	30	400	0.3	33.65	29.93	57	0.01	48	0.06	37	0.4	31	3.62
san400_0.7_3.clq	22	400	0.3	22.00	53.75	40	0.01	41	0.06	28	0.38	26	4.53
san400_0.9_1.clq	100	400	0.1	100.00	3.3	203	0.01	147	0.08	120	0.7	109	5.95
sanr200_0.7.clq	18	200	0.3	33.84	2.13	67	0	43	0.01	33	0.09	27	0.95
sanr200_0.9.clq	42	200	0.1	59.89	0.47	97	0.01	72	0.02	66	0.17	60	1.52
sanr400_0.5.clq	13	400	0.5	38.61	66.02	96	0.01	49	0.05	32	0.24	23	4.24
sanr400_0.7.clq	21	400	0.3	58.49	30.62	119	0.01	78	0.06	59	0.36	46	4.43

References

[1] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27:509–516, 1978.

[2] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2007.

[3] B. Becker, M. Behle, F. Eisenbrand, and R. Wimmer. BDDs in a branch and cut framework. In S. Nikoletseas, editor, *Experimental and Efficient Algorithms, Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA 05)*, volume 3503 of *Lecture Notes in Computer Science*, pages 452–463. Springer, 2005.

[4] Markus Behle and Friedrich Eisenbrand. 0/1 vertex and facet enumeration with BDDs. In *ALENEX*. SIAM, 2007.

[5] David Bergman, Andre A. Cire, W.-J. van Hoeve, and J. N. Hooker. Variable ordering for the application of BDDs to the maximum independent set problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2012)*. Springer, to appear.

[6] David Bergman, Willem Jan van Hoeve, and John N. Hooker. Manipulating MDD relaxations for combinatorial optimization. In Tobias Achterberg and J. Christopher Beck, editors, *CPAIOR*, volume 6697 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2011.

[7] Bollig and Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEETC: IEEE Transactions on Computers*, 45, 1996.

[8] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

[9] Rudiger Ebendt, Wolfgang Gunther, and Rolf Drechsler. An improved branch and bound algorithm for exact BDD minimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(12):1657–1663, 2003.

[10] T. Hadzic and J. N. Hooker. Postoptimality analysis for integer programming using binary decision diagrams, presented at GICOLAG workshop (Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geometry), Vienna. Technical report, Carnegie Mellon University, 2006.

[11] T. Hadzic and J. N. Hooker. Cost-bounded binary decision diagrams for 0-1 programming. In E. Loute and L. Wolsey, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial*

Optimization Problems (CPAIOR 2007), volume 4510 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2007.

[12] T. Hadzic, J. N. Hooker, B. O'Sullivan, and P. Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In P. J. Stuckey, editor, *Principles and Practice of Constraint Programming (CP 2008)*, volume 5202 of *Lecture Notes in Computer Science*, pages 448–462. Springer, 2008.

[13] S. Hoda, W.-J. van Hoeve, and John N. Hooker. A systematic approach to MDD-based constraint programming. In *Proceedings of the 16th International Conference on Principles and Practices of Constraint Programming*, Lecture Notes in Computer Science. Springer, 2010.

[14] Alan John Hu. Techniques for efficient formal verification using binary decision diagrams. Thesis CS-TR-95-1561, Stanford University, Department of Computer Science, December 1995.

[15] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal*, 38:985–999, 1959.

[16] Elsa Loekito, James Bailey, and Jian Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowl. Inf. Syst*, 24(2):235–268, 2010.

[17] I. Wegener. *Branching programs and binary decision diagrams: theory and applications.* SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, 2000.