2011

Manipulating MDD Relaxations for Combinatorial Optimization

David Bergman
Carnegie Mellon University

Willem-Jan van Hoeve
Carnegie Mellon University, vanhoeve@andrew.cmu.edu

John N. Hooker
Carnegie Mellon University, john@hooker.tepper.cmu.edu

Recommended Citation
T. Achterberg and J.C. Beck (Eds.): CPAIOR 2011, LNCS 6697, 20- 35.

Manipulating MDD Relaxations for Combinatorial Optimization

David Bergman, Willem-Jan van Hoeve, J. N. Hooker

Tepper School of Business, Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213, U.S.A.
{dbergman,vanhoeve}@andrew.cmu.edu, john@hooker.tepper.cmu.edu

Abstract. We study the application of limited-width MDDs (multi-valued decision diagrams) as discrete relaxations for combinatorial optimization problems. These relaxations are used for the purpose of generating lower bounds. We introduce a new compilation method for constructing such MDDs, as well as algorithms that manipulate the MDDs to obtain stronger relaxations and hence provide stronger lower bounds. We apply our methodology to set covering problems, and evaluate the strength of MDD relaxations to relaxations based on linear programming. Our experimental results indicate that the MDD relaxation is particularly effective on structured problems, being able to outperform state-of-the-art integer programming technology by several orders of magnitude.

1 Introduction

Binary Decision Diagrams (BDDs) [1, 19, 6] provide compact graphical representations of Boolean functions, and have traditionally been used for circuit design and formal verification [17, 19]. More recently, however, BDDs and their generalization Multivalued Decision Diagrams (MDDs) [18] have been used in Operations Research for a variety of purposes, including cut generation [3], vertex enumeration [5], and post-optimality analysis [12, 13].

In this paper, we examine the use of BDDs and MDDs as relaxations for combinatorial optimization problems. Relaxation MDDs were introduced in [2] as a replacement for the domain store relaxation, i.e., the Cartesian product of the variable domains, that is typically used in Constraint Programming (CP). MDDs provide a richer data structure that can capture a tighter relaxation of the feasible set of solutions, as compared with the domain store relaxation. In order to make this approach scalable, MDD relaxations of limited size are applied. Various methods for compiling these discrete relaxations are provided in [14]. The methods described in that paper focus on iterative splitting and edge filtering algorithms that are used to tighten the relaxations. Similar to classical domain propagation, such MDD propagation algorithms have been developed for individual (global) constraints, including inequality constraints, equality constraints, *alldifferent* constraints and *among* constraints [14, 15].

The focus of the current work is the application of limited-width MDD relaxations in the context of optimization problems. We explore two main topics.

Firstly, we investigate a new method for building approximate MDDs. We introduce a top-down compilation method based on approximating the set of completions of partially assigned solutions. This procedure differs substantially from the ideas in [2] in that we do not compile the relaxation by splitting vertices, but by merging vertices when the size of the partially constructed MDD grows too large.

Secondly, and more specific to optimization, we introduce a method to improve the lower bound provided by an MDD relaxation. It is somewhat parallel to a cutting plane algorithm in that it "cuts off" infeasible solutions so as to tighten the bound. Unlike cutting planes, however, it can begin with any valid lower bound, perhaps obtained by another method, and tighten it. The bound becomes tighter as more time is invested.

The resulting mechanism is a pure inference algorithm that can be used analogously to a pure cutting plane algorithm. We envision, however, that MDD relaxations would be most profitably used as a bounding technique in conjunction with a branch-and-bound search, much as separation algorithms are used in integer programming. Nonetheless we find in this paper that, even as a pure inference algorithm, MDD relaxation can outperform state-of-the-art integer programming technology on specially structured instances.

One advantage of an MDD relaxation is that it is always easy to solve (as a shortest path problem) whether the original problem is linear or nonlinear. This suggests that MDDs might be most competitive on nonlinear discrete problems. Nonetheless we deliberately put MDDs at a competitive disadvantage by applying them to a problem with linear inequality constraints—namely, to the set covering problem, which is well suited to integer programming methods.

We compare the strength of bounds provided by MDDs with those provided by the linear programming relaxation and cutting planes. We also compare the speed with which MDDs (used as a pure inference method) and integer programming solve the problem. We find that MDDs are much superior to conventional integer programming when the ones in the constraint matrix lie in a relatively narrow band. That is, the matrix has relatively small bandwidth, meaning that the maximum distance between any two ones in the same row is limited.

The bandwidth of a set covering matrix can often be reduced, perhaps significantly, by reordering the columns. Thus MDDs can solve a given set covering problem much more rapidly than integer programming if its variables can be permuted to result in a relatively narrow bandwidth. Algorithms and heuristics for minimum bandwidth ordering are discussed in [20, 7–9, 11, 21–23].

The remainder of the paper is organized as follows. In Section 2 we define MDDs more formally and introduce notation. In Section 3 we describe a new top-down compilation method for creating relaxation MDDs. In Section 4 we present our value enumeration scheme to produce lower bounds. In Section 5 we discuss applying the ideas of the paper to set covering problems. In Section 6 we report on experiments results where we apply the ideas of the paper to set covering problems. We conclude in Section 7.

2 Preliminaries

In this work a *Multivalued Decision Diagram* (MDD) is a layered directed acyclic multi-graph whose nodes are arranged in $n+1$ layers, $L_1, L_2, \ldots, L_{n+1}$. Layers L_1 and L_{n+1} consist of single nodes; the root r and the terminal t, respectively. All arcs in the MDD are directed from nodes in layer j to nodes in layer $j+1$.

In the context of Constraint Satisfaction Problems (CSPs) or Constraint Optimization Problems (COPs), we use MDDs to represent assignments of values to variables. A CSP is specified by a set of constraints $C = \{C_1, C_2, \ldots C_m\}$ on a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ with respective finite domains $D_1, \ldots, D_n$, and a COP is specified by a CSP together with an objective function f to be minimized. By a *solution* to a CSP (COP) we mean an assignment of values to variables where the values assigned to the variables appear in their respective domains. By a *feasible solution* we mean a solution that satisfies each of the constraints in C, and the *feasible set* is the set of all feasible solutions. For a COP, an *optimal* solution is a feasible solution x^* such that for any other feasible solution $\tilde{x}, f(x^*) \leq f(\tilde{x})$.

We use MDDs to represent a set of solutions to a CSP, or COP, as follows. We let the layers $L_1, \ldots, L_n$ correspond to the problem variables $x_1, \ldots, x_n$, respectively. Node $u \in L_j$ has label $\text{var}(u) = j$, representing its variable index. Arc (u, v) with $\text{var}(u) = j$ is labeled with *arc domain* $d_{u,v}$, by an element of the domain of variable x_j, i.e., $d_{u,v} \in D_j$. All arcs directed out of a node must have distinct labels.

A path p from node u_i to node u_k, $i < k$, along arcs $a_i, a_{i+1}, \ldots, a_{k-1}$ corresponds to the assignment of the values d_{a_j} to the variables x_j, for $j = i, i+1, \ldots, k-1$. In particular, we see that any path from the root r to the terminal $t, p = (a_1, \ldots, a_n)$, corresponds to the solution x^p, where $x^p_j = d_{a_j}$. We note that as an MDD is a multi-graph, two paths p_1, p_2, along nodes $r = u_1, \ldots, u_n, t$ may correspond to multiple solutions as there may be multiple arcs from u_j to u_{j+1} corresponding to different assignments of values to the variable x_j.

The set of solutions represented by MDD M is $\text{Sol}(M) = \{x^p | p \in P\}$ where P is the set of paths from r to t. The *width* of layer L_j is given by $\omega_j = |L_j|$, and the *width* of MDD M is given by $\omega(M) = \max_{j \in \{1,2,\ldots,n\}} \omega_j$. The *size* of M is denoted by $|M|$, the number of nodes in M.

For a given CSP $\mathcal{P}$, let $X(\mathcal{P})$ be the set of feasible solutions for $\mathcal{P}$. An *exact* MDD M for $\mathcal{P}$ is any MDD for which $\text{Sol}(M) = X(\mathcal{P})$. A *relaxation* MDD M_{rel} for $\mathcal{P}$ is any MDD for which $\text{Sol}(M_{\text{rel}}) \supseteq X(\mathcal{P})$. For the purposes of this paper, relaxation MDDs are of limited width, in that we require that $\omega_j \leq W$, for some predefined W. This ensures that the relaxation has limited size which is necessary since even for single constrained problems, the feasible set may correspond to an MDD of exponential size (for example inequality constrained problems [4]).

Finally, we note that for a large class of objective functions (e.g., for separable functions), optimizing over the solutions represented by an MDD corresponds to finding a shortest path in the MDD [2]. For example, given a linear objective function $\min cx$, we associate with each arc (u, v) in the MDD a *cost* $c(u, v)$,

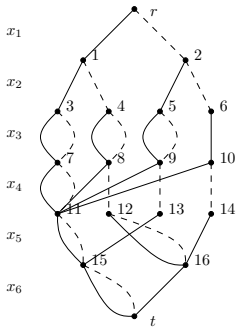


Fig. 1. Exact MDD for Example 1.

where $c(u, v) = c_{\text{var}(u)} \cdot d_{u,v}$. Then it is clear that a shortest path from r to t corresponds to the lowest cost solution represented by the MDD.

Example 1. As an illustration, consider the CSP consisting of binary variables $x_1, x_2, \ldots, x_6$, and constraints

$$\begin{aligned} C_1: &\quad x_1 + x_2 + x_3 \geq 1, \\ C_2: &\quad x_1 + x_4 + x_5 \geq 1, \\ C_3: &\quad x_2 + x_4 + x_6 \geq 1. \end{aligned}$$

An exact MDD representation of the feasible set is given in Fig. 1, where arc (u, v) being solid/dashed corresponds to the arc setting $\text{var}(u)$ to 1/0.

3 Top-Down MDD Compilation

As discussed above, there are several methods that can be used to construct both exact and approximate MDDs. In this section we propose a new top-down method for creating approximate MDDs.

3.1 Exact Top-Down Compilation

We first discuss an exact top-down compilation method, which is based on the notion of *node equivalence*.

Given a path p from r to u, let $F(p)$ be the set of feasible completions of the corresponding partial assignment. That is, if $(x_1, \ldots, x_k) = (d_1, \cdots, d_k) = d$ is the partial assignment represented by p, then $F(p) = \{y \in D_{k+1} \times \cdots \times D_n | (d, y) \text{ is feasible } \}$. We say that two paths p, p' from r to the same layer are said to be *equivalent* if $F(p) = F(p')$.

Analogously, we define $F(u)$ to be the set of completions at node u, so that $F(u) = \bigcup_{p \in P} F(p)$, where P is the set of paths from r to u. We note that in an exact MDD all paths terminating at a node u are equivalent.

A *node equivalence test* determines when two nodes u, u' on the same layer have the same set of feasible completions. In other words, this test determines when $F(u) = F(u')$. Testing whether two nodes have the same set of feasible completions requires maintaining a *state* I_v at each node [15]. The state of node u should contain all facts about the paths ending at v to run an equivalence test. In addition, it is useful to know when a partial assignment cannot be completed to a feasible solution for a CSP. In such a case, we let the state of such a path, or more generally a node, be $\hat{0}$, to signal that there are no completions of this path/node.

Now, using a properly defined node equivalence test, one can create an exact MDD using Algorithm 1. Given that layers $L_1, \ldots, L_j$ have been created, we examine the nodes in L_j one by one. When examining node u, for each domain value $d \in D_j$ we calculate the new state I_{new} that results from adding $x_j = d$ to the partial paths ending at u. If no other nodes on layer L_{j+1} have the same state (i.e. the same set of feasible completions) we add a new node v to L_{j+1} and the arc (u, v) with arc domain d, and set $I_v = I_{\text{new}}$. If however there is some node $w \in L_{j+1}$ with $I_w = I_{new}$ we know that all paths starting at r, ending at u and having $x_j = d$ will have the same set of feasible completions as w. Therefore, we simply add the arc (u, w) with arc domain d.

Algorithm 1 Top-Down MDD Compilation

```
1: L_1 = {r}
2: for j = 1 to n do
3:    L_{j+1} = ∅
4:    for all u ∈ L_j do
5:       for all d ∈ D(x_j) do
6:          calculate I_new, the state for all paths starting at r, ending at u, and including
            x_j = d
7:          if I_new ≠ 0̂ then
8:             if there exists w ∈ L_{j+1} with I_w = I_new then
9:                add arc (u, w) with d_{u,w} = d
10:            else
11:               add node v to L_{j+1}
12:               add arc (u, v) with d_{u,v} = d
13:               set I_v = I_new
14:            end if
15:         end if
16:      end for
17:   end for
18: end for
```

We will be modifying Algorithm 1 later to create approximate MDDs. First, however, we discuss specific exact MDDs for the feasible set *satisfying a single equality constraint*. Such MDDs will be applied in our value enumeration method for tightening lower bounds, presented in Section 4.

Lemma 1. *Let $\mathcal{P}$ be a CSP on n binary variables with the single constraint $\sum_{j=1}^{n} c_j x_j = c$, for a given integer c, and integer coefficients $c_j \geq 0$. An exact MDD for $\mathcal{P}$ has maximum width $c+1$.*

Proof. We apply Algorithm 1. Given a node u, let p be any path from r to u, and let $a_1, \ldots, a_k$ be the arcs along this path, which set variables $x_1, \ldots, x_k$ to the arc domain values $d_{a_1}, \ldots, d_{a_k}$. We define $I_u = \sum_{j=1}^{k} c_j \cdot d_{a_j}$. Using this label as the state of node u we see that two nodes u and v have the same set of feasible completions if and only if $I_u = I_v$. In addition, if $I_w \geq c+1$ for some node w, it is clear that all paths from r to w have no feasible completions. Therefore we can have at most $c+1$ nodes on any layer. □

We note that Lemma 1 is very similar to the classical pseudo-polynomial characterization of knapsack constraints.

3.2 Approximate Top-Down Compilation

In general, an exact MDD representation of all feasible solutions to a CSP may be of exponential size, and therefore generating exact MDDs for combinatorial optimization problems is not practical. In light of this we use relaxation MDDs to approximate the set of feasible solutions. In this section we outline one possible method for generating approximate MDDs, by modifying Algorithm 1.

In order to create a relaxation MDD we merge nodes during the top-down compilation method presented in Algorithm 1 when the width of layer j exceeds a certain preset maximum allotted width W. To accomplish this, we select two nodes and modify their states in a relaxed fashion, ensuring that all feasible solutions will remain in the MDD when it is completed. More formally, if we select nodes u_1 and u_2 to merge, we need to modify their states I_{u_1}, I_{u_2} in such a way as to make them equivalent with respect to the equivalence test used to merge nodes during the top-down compilation. We define a certain *relaxation operation* $\oplus$ on the state of nodes as follows.[1] If for nodes u_1 and u_2 we change their associated states to $I(u_1) \oplus I(u_2)$, any feasible completion of the paths from the root to u_1 and u_2 will remain when the terminal is reached. This is outlined in Algorithm 2, which is to be inserted between lines 17 and 18 in Algorithm 1. In Section 5 we describe such an operation in detail, for set covering problems.

The quality of the relaxation MDD generated using the modification of Algorithm 1 hinges largely on the method used for selecting two nodes to combine. We propose several heuristics for this choice in the following table:

Name	Node selection method
H_1	select u_1, u_2 uniformly at random among all pairs in S_{j+1}
H_2	select u_1, u_2 such that $f(u_1), f(u_2) \geq f(v), \forall v \in S_{j+1}, v \neq u_1, u_2$
H_3	select u_1, u_2 such that I_{u_1} and I_{u_2} are *closest* among all pairs in S_{j+1}

[1] Here we follow the notation $\oplus$ that was used in [15] for their analogous operation for aggregating node information.

Algorithm 2 Top-Down Relaxation Compilation

1: **while** $|S_{j+1}| > W$ **do**
2: **select** nodes $u_1, u_2 \in S_{j+1}$
3: create node u
4: for every arc directed at u_1 or u_2 redirect arc to u with the same arc domain
5: $I(u) = I(u_1) \oplus I(u_2)$
6: $S_{j+1} \leftarrow S_{j+1} \backslash \{u_1, u_2\} \cup \{u\}$
7: **end while**

The rationale behind each of the methods are the following. Method H_1 calls for randomly choosing which nodes to combine. Randomness often helps in combinatorial optimization and applying it in this context may work as well. H_2 combines nodes that have the greatest shortest path lengths. For this we let $f(u)$ be the shortest path length from the root to u in the partially constructed MDD. Choosing such a pair of nodes allows for approximating the set of feasible solutions in parts of the MDD where the optimal solution is unlikely to lie, and retaining the exact paths in sections of the MDD where the optimal solution is likely to lie. H_3 combines nodes that have similar states. For particular types of states and equivalence tests, we must determine the notion of *closest*. This method is sensible because these nodes will most likely have similar sets of completions, allowing the relaxation to better capture the set of feasible solutions.

4 Value Enumeration

We next discuss the application of MDD relaxations for obtaining lower bounds on the objective function, in the context of COPs. We propose to obtain and strengthen these bounds by means of successive value enumeration. Value enumeration is a method that can be used to increase any lower bound on a COP via a relaxation MDD.

Suppose we have generated a relaxation MDD M_{rel}. We then generate an MDD representing every solution in M_{rel} with objective function value equal to the best lower bound. There are several ways to accomplish this, but in general this MDD can have exponential size. However, for some important cases the MDD representing every solution equal to a particular value has polynomial size.

For example, suppose we have a COP with objective function equal to the sum of the variables, i.e., $f(x) = \sum_{j=1}^{n} x_j$, where we assume that the variable domains are integral. Given a lower bound z_{LB}, the reduced MDD for the set of solutions with objective value equal to z_{LB}, $M_{z_{LB}}$, has width $\omega(M_{z_{LB}}) = z_{LB}+1$, by Lemma 1. The same holds for other linear objective functions as well.

In any case, suppose we have the desired MDD $M_{z_{LB}}$, where $\mathrm{Sol}(M_{z_{LB}})$ is the set of all solutions with objective value equal to z_{LB}. Now, consider the set of solutions $S = \mathrm{Sol}(M_{z_{LB}}) \cap \mathrm{Sol}(M_{\mathrm{rel}})$. As this is the intersection between the solutions represented by the relaxation and every solution equal to the lower

bound z_{LB}, showing that there is no feasible solution in S allows us to increase the lower bound.

Constructing an MDD M representing the set of solutions $S = \text{Sol}(M_{z_{LB}}) \cap \text{Sol}(M_{\text{rel}})$ can be done in time $\mathcal{O}(|M_{z_{LB}}| \cdot |M_{\text{rel}}|)$ and has maximum width $\omega(M) \leq \omega(M_{z_{LB}}) \cdot \omega(M_{\text{rel}})$ [6]. As the width of $M_{z_{LB}}$ has polynomial size for certain objective functions and the width of M_{rel} is bounded by some preset W, the width of the resulting MDD will not grow too large in these cases.

The value enumeration scheme proceeds by enumerating all of the solutions in M. If we find a feasible solution, we have found a witness for our lower bounds. Otherwise, we can increase the lower bound by 1. Of course, this method is only practical if we can enumerate these paths efficiently.

Observe that we do not need to start the value enumeration scheme with the value of the shortest path in the original MDD. In fact, we can start with any lower bound. For example, we can use LP to find a strong lower bound and then apply this procedure to any relaxation MDD.

As described above, in order to increase the bound, we are required to certify that none of the paths in M correspond to feasible solutions. Of course this can be done by a naive enumeration of all of the paths in M. However, we use MDD-based CP, as described in [2], in unison with a branching procedure to certify this. In particular we apply MDD filtering algorithms to reduce the size of the MDD $M_{z_{LB}}$, based on the constraints that constitute the COP. In Section 5.3 we will describe a new MDD filtering algorithm that we apply to set covering problems.

5 Application to Set Covering

In this section we describe how to apply the ideas of the paper to set covering problems. We describe a node equivalence test and the state that is necessary to carry out the test. We also describe the operation $\oplus$ that can be used to change the states of the nodes so that we can generate relaxation MDDs.

5.1 Equivalence Test

The well-studied set covering problem is a COP with n binary variables and m constraints, each on a subset C_i of the variables, which require that $\sum_{j \in C_i} x_j \geq 1, i \in \{1, \ldots, m\}$. The objective is to minimize the sum of the variables (or a weighted sum).

The first step in applying our top-down compilation method is defining an equivalence test between partial assignments of values to variables. For set covering problems we do this by equating a set covering instance with its equivalent logic formula. Each constraint C_i can be viewed as a clause $\vee_{j \in C_i} x_j$ and the set covering problem is equivalent to satisfying $F = \bigwedge_i \vee_{j \in C_i} x_j$.

Using this interpretation of set covering problems, one can develop a complete equivalence test by removing clauses that are implied by other clauses. Clause C *absorbs* clause D if all of the literals of C are contained in D. In such a

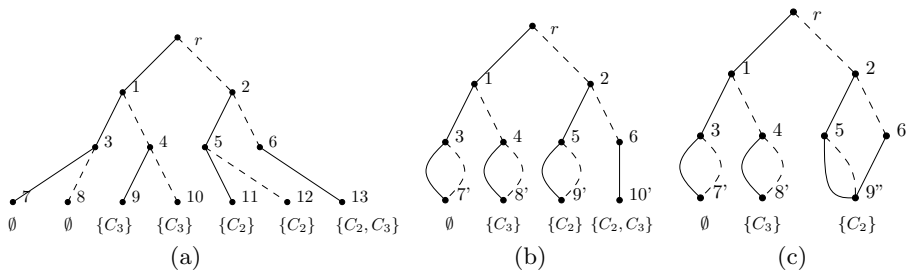


Fig. 2. (a) Exact MDD before combining nodes with the same state, (b) Exact MDD after combining nodes with the same state, (c) MDD after merging node 9' and 10' into 9", making a partially constructed relaxation.

case, satisfying clause C implies that clause D will be satisfied. As an example, consider the two clauses $C = (x_1 \vee x_2)$ and $D = (x_1 \vee x_2 \vee x_3)$. It is clear that if some literal in C is set to `true` then clause D will be satisfied.

Therefore, to develop the equivalence test, for any partial assignment x we delete any clause C_i for which there exists a variable in the clause that is already set to 1, and then delete all absorbed clauses, resulting in the logical formula $F(x)$. We let I_x be the set of clauses which remain in $F(x)$. Doing so ensures that two partial assignments x^1 and x^2, will have the same set of feasible completions if and only if $I_{x^1} = I_{x^2}$. Note that since each literal is positive in all clauses of a set covering instance, this test can be performed in polynomial time [16].

To create an exact MDD for a set covering instance (using Algorithm 1), we let the state I_u at node u be equal to I_x for the partial assignment given by the arc domains on all paths from the root to u. Two nodes u and v will have the same set of feasible completions if and only if $I_u = I_v$, and so the node equivalence test simply compares I_u with I_v.

Example 2. Continuing Example 1, we interpret the constraints C_1, C_2, and C_3 as set covering constraints. In Fig. 2(a) we see the result of applying the top-down compilation algorithm (following the variable order $x_1, x_2, \ldots, x_6$) and never combining nodes based on their associated states , for the first three layers of the MDD. Below the bottom nodes, we depict the states of the partially constructed paths ending at those nodes. For example, along this path $(r, 2, 5, 11)$, variables x_2 and x_3 are set to 1. Therefore, constraints C_1 and C_3 are satisfied for any possible completion of this path, and so the state at node 11 is C_2. Since node 11 and node 12 have the same state, we can combine these nodes into node 9', as shown in Fig. 2(b). Similarly, nodes 7 and 8 are combined into node 7' and nodes 9 and 10 are combined into node 8'.

5.2 Relaxation Operation

We next discuss our relaxation operator $\oplus$ that is applied to merge two nodes in a layer. For set covering problems, we let $\oplus$ represent the typical set intersection.

As an illustration, for the instance in Example 2, suppose we decided that we want to decrease the width of layer 4 by 1. We would select two nodes (in Fig. 2(b) we select nodes 9' and 10') and combine them (making node 9" as seen in Fig. 2(c)), modifying their states to ensure that all feasible paths remain upon completing the MDD. Notice that by taking the intersection of the states of the nodes 9' and 10' we now label 9" with C_2. Before merging the nodes, all partial paths ending at node 10' needed a variable in both constraint C_2 and C_3 to be set to 1. After taking the intersection, we are relaxing this condition, and only require that for all partial paths ending at 9", all completions of this path will set some variable in constraint C_2 to 1, ignoring that this needs to also happen for constraint C_3.

5.3 Filtering

As discussed above, during the value enumeration procedure, it is desirable to perform some MDD filtering to decrease the search space. This filtering can be applied to arc domain values, as described in [2], but also to the states represented in the nodes themselves, as we will describe here in the context of set covering problems.

We associate two 0/1 m-dimensional state variables, $s(v), z(v)$, to each node v in the MDD. The value $s(v)_i$ will be 1 if for all paths from the root to v, there is no variable in constraint C_i which is set to 1. Similarly, $z(v)_i$ will be 1 if for all paths from v to the terminal, there is no variable in C_i which is set to 1.

Finding the values $s(v)_i, z(v)_i$ is easily accomplished by the following simple algorithm. Start with $s(r)_i = 1$ for all i. Now, let node v have parents $u_1, u_2, \ldots, u_k$, and each edge (u_p, u) fixes variable x_j to value $v_p \in \{0, 1\}$. Then,

$$s(v)_i = \prod_{p=1}^{k} s'(u_p)_i,$$

where

$$s'(u_p)_i = \begin{cases} 0 & \text{if } x_j \in C_i \text{ and } v_p = 1 \\ s(u_p)_i & \text{otherwise} \end{cases}$$

The values $z(v)_i$ are calculated in the same fashion, except switching the direction of all arcs in the MDD and starting with $z(t)_i = 1$, where t is the terminal of the MDD.

A node v can now be eliminated whenever there is an index i such that $s(v)_i = z(v)_i = 1$. This is because for all paths from r to v there is no variable in C_i set to 1, and on all paths from v to t, there is no variable in C_i set to 1.

We note here that as in domain store filtering, certain propagators for MDDs are *idempotent*, in that reapplying the filtering algorithm with no additional changes results in no more filtering. The filtering algorithm presented here is not idempotent, i.e, applying it multiple times could result in additional filtering. In our computational experiments we address how this impacts the efficiency of the overall method.

6 Experimental Results

In this section, we present experimental results on randomly generated set covering instances. Our results provide evidence that relaxations based on MDDs perform well when the constraint matrix of a set covering instance has a small bandwidth. We test this by generating random set covering instances with varying bandwidths and comparing solution times via pure-IP (using CPLEX), pure-MDD, and a hybrid MDD-IP method.

In all of the reported results, unless specified otherwise, we apply our MDD-based algorithm until it finds a feasible solution. That is, we solve these set covering problems by continuously improving the relaxation through our value enumeration scheme until we find a feasible (optimum) solution.

6.1 Bandwidth and the Minimum Bandwidth Problem

The *bandwidth* of a matrix A is defined as

$$b_w(A) = \max_{i \in \{1,2,\ldots,m\}} \{ \max_{j,k: a_{i,j}, a_{i,k} = 1} \{j - k\} \}.$$

The bandwidth represents the largest distance, in the variable ordering given by the constraint matrix, between any two variables that share a constraint. The smaller the bandwidth, the more structured the problem, in that the variables participating in common constraints are close to each other in the ordering. The *minimum bandwidth problem* seeks to find a variable ordering that minimizes the bandwidth [20, 7–9, 11, 21–23]. This underlying structure, when present in A, can be captured by MDDs and results in good computational performance.

6.2 Problem Generation

To test the statement that MDD based relaxations provide strong relaxations for structured problems, we generate set covering instances with a fixed constraint matrix density d (the number of ones in the matrix divided by $n \cdot m$) and vary the bandwidth b_w of the constraint matrix.

We generate random instances with a fixed number of variables n, constraint matrix density d, and bandwidth b_w, where each row i has exactly $k = d \cdot n$ ones. For constraint i the k ones are chosen uniformly at random from variables $x_{i+1}, x_{i+2}, \ldots, x_{i+b_w}$ As an example, a constraint matrix with $n = 9, d = \frac{1}{3}$ and $b_w = 4$ may look like

$$A = \begin{pmatrix} 1\,1\,0\,1\,0\,0\,0\,0\,0 \\ 0\,1\,1\,1\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,1\,1\,0\,0\,0 \\ 0\,0\,0\,1\,0\,1\,1\,0\,0 \\ 0\,0\,0\,0\,1\,0\,1\,1\,0 \\ 0\,0\,0\,0\,0\,0\,1\,1\,1 \end{pmatrix}$$

As b_w grows, the underlying staircase-like structure of the instances dissolves. Hence, by increasing b_w, we are able to test the impact of the structure in the set covering instances on our MDD-based approach.

Consider the case when $b_w = k$. For such problems, as A is totally unimodular [10], the LP optimal solution will be integral, and so the corresponding IP will solve the problem at the root node. Similarly, we show here that the set of feasible solutions can be exactly represented by an MDD with width bounded by $m+1$. In particular, for any node u created during the top-down compilation method, I_u must be of the form $(0, 0, \ldots, 0, 1, 1, \ldots, 1)$. This is because, given any partial assignment fixing the top j variables, if some variable in constraint C_i is fixed to 1, then for any constraint C_k, with $k \leq i$, there must be some variable also fixed to 1. Hence, $\omega(M) \leq m+1$. Therefore, such problems are also easily handled by MDD-based methods. Increasing the bandwidth, however, destroys the totally unimodular property of A and the bounded width of M. Therefore, increasing the bandwidth allows us to test how sensitive the LP and the relaxation MDDs are to changes in the structure of A.

6.3 Evaluating the MDD Parameters

In Section 3.2 we presented three possible heuristics for selecting nodes to merge. In preliminary computational tests, we found that using the heuristic based on shortest partial path lengths, H_2, seemed to provide the strongest MDD relaxations, and so we employ this heuristic.

The next parameter that must be fixed is the preset maximum width W that we allow for the MDD relaxations. Each problem (and even more broadly for each application of MDD relaxations to CSP/COPs) has a different optimal width. To test for an appropriate width for this class of problems, we generate 100 instances with $n = 100, k = 20$ and $b_w = 35$.

In Figure 3(a) we report the average solution time, over the 100 instances, for different maximum allowed widths W. Near $W = 35$ we see the fastest solution times, and hence for the remainder of the experimental testing we fix W at 35. We note here that during our preliminary computational tests, the range of widths that seemed to perform best was $W \in [20, 40]$.

Another parameter of interest is the number of times we allow the filtering algorithm to run before branching. As discussed in Section 5.3 the filtering algorithm presented above for set covering problems is not idempotent and applying the filtering once or for several rounds has different impacts on the solution time. In Figure 3(b) we report solution time versus the number of rounds of filtering averaged over the 100 instances with $W = 35$. Applying the filtering algorithm once yielded the fastest solution times and so we use this for the remainder of the experiments.

6.4 Evaluating the Impact of the Bandwidth

Next we compare the performance of our MDD-based approach with IP. We also compare the performance of these two methods with a hybrid MDD/IP

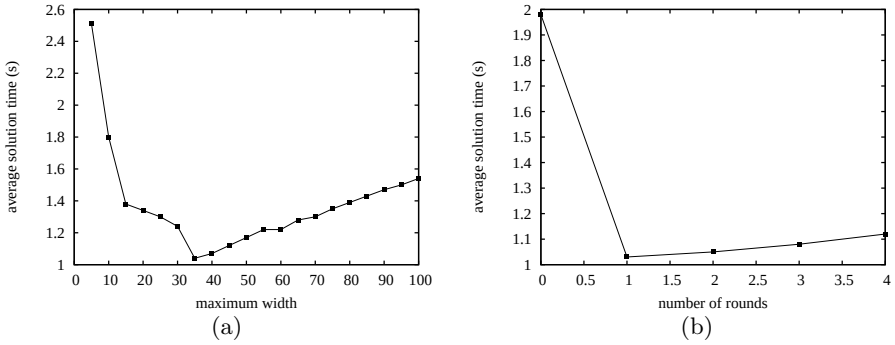


Fig. 3. (a) Maximum width W vs. solution time, (b) Number of rounds of filtering vs. solution time.

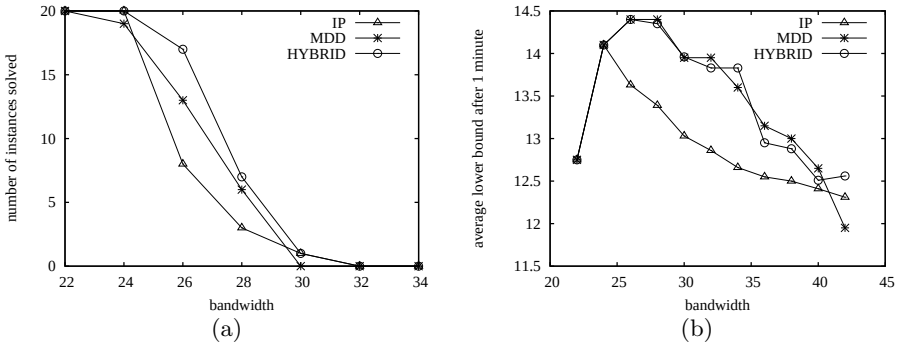
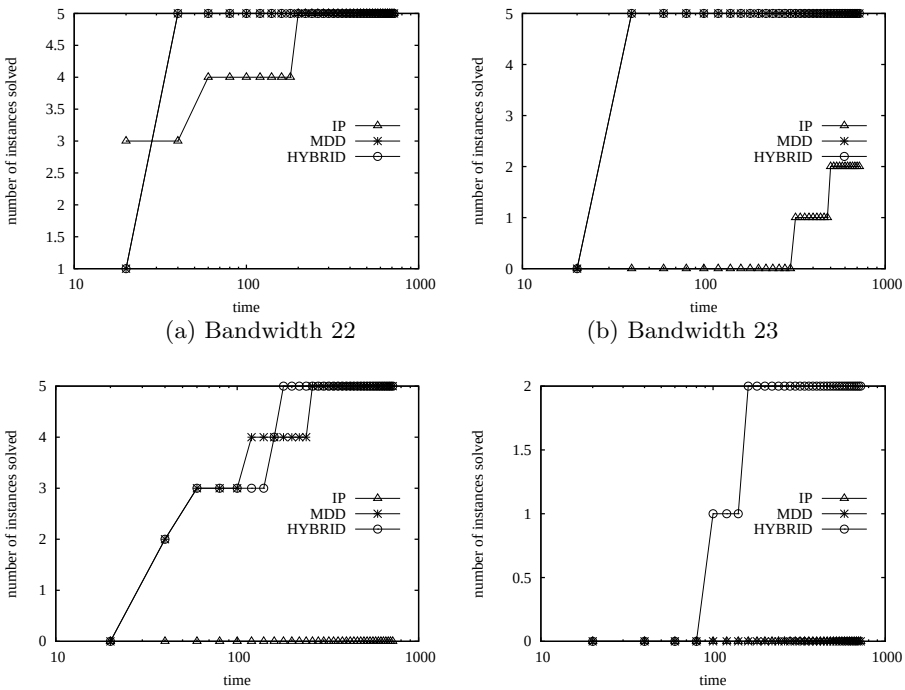


Fig. 4. (a) Number of instances solved in 1 minute for different bandwidths, (b) Average lower bound in 1 minutes for different bandwidths.

approach. For the hybrid method, the MDD algorithm runs for a fixed amount of time and then passes the lower bound on the objective function to IP as a initial lower bound on the objective function.

We report results for random instances with $n = 250$, $k = 20$ and bandwidth $b_w \in \{22, 24, \ldots, 44\}$ (20 instances per configuration). In Figure 4(a) we show, for increasing bandwidths, the number of instances solved in 60 seconds using the three proposed methods. For the hybrid method, we let the MDD method run for 10 seconds, and then pass the bound z_{LB} given by the MDD method to the IP and let the IP solver run for an additional 50 seconds. In addition, in Figure 4(b) we show, for increasing bandwidths, the best lower bound provided by the three methods after one minute.

For the lower bandwidths, we see that both the MDD-based approach and the hybrid approach outperform IP, with the hybrid method edging out the pure MDD method. As the bandwidth grows, however, the underlying structure that the MDD is able to exploit dissolves, but still the hybrid approach performs best.



(a) Bandwidth 22 (b) Bandwidth 23

(c) Bandwidth 24 (d) Bandwidth 25

Fig. 5. Performance profile for pure-IP, pure-MDD, and hybrid MDD/IP for instances width various bandwidth. Time is reported in log-scale.

6.5 Scaling Up

Here we present results on instances with 500 variables, and again with $k = 20$, to evaluate how the algorithms scale up. We have generated instances for various bandwidths b_w between 21 and 50 (5 random instances per configuration), and we report the most interesting results corresponding to the 'phase transition', i.e., $b_w \in \{22, 23, 24, 25\}$. We compare the three solution methods, allowing the algorithms to run for 12 minutes.

In the four plots given in Figure 5, we depict the performance profile of the three methods for the different bandwidths. We show for each bandwidth the number of instances solved by time t. As the bandwidth increases, we see that the IP is unable to solve many of the instances that the MDD-based method can, and for $b_w = 25$, neither the pure-IP nor the pure-MDD based methods can solve the instances, while the hybrid method was able to solve 2 of the 5 instances.

Figure 6(a) displays the lower bound given by the three approaches versus time, averaged over the 5 instances. We run the algorithms for 5 minutes and see that the lower bound given by the MDD-based approach dominates the IP bound, especially at small bandwidths. However, as the bandwidth grows, as

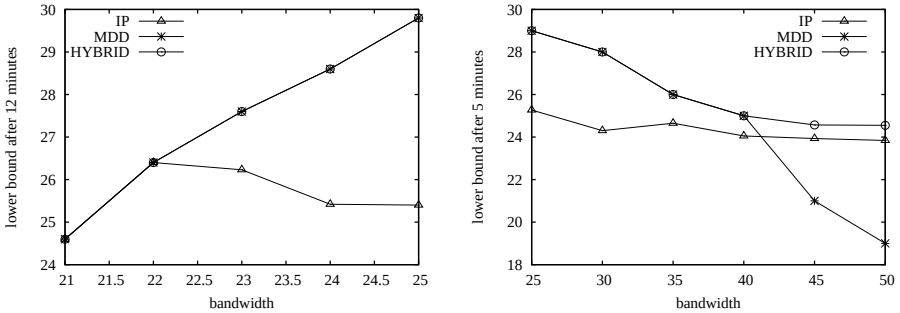


Fig. 6. (a) Bandwidth versus lower bound (12 minute time limit), (b) Larger bandwidths versus lower bound (5 minute time limit).

shown in Figure 6(b), the structure captured by the relaxation MDDs no longer exists and the pure-IP method is able to find better bounds. However, even at the larger bandwidths, the hybrid method provides the best bounds.

7 Conclusion

In conclusion, we have examined how relaxation MDDs can help in providing lower bounds for combinatorial optimization problems. We discuss methods for providing lower bounds via relaxation MDDs and provide computational results on applying these ideas to randomly generated set covering problems. We show that in general we can quickly improve upon LP bounds, and even outperform state-of-the-art integer programming technology on problem instances for which the bandwidth of the constraint matrix is limited. Finally, we have shown how a hybrid combination of IP and MDD-based relaxation can be even more effective.

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 *Proceedings of CP*, volume 4741 of *LNCS*, pages 118–132. Springer, 2007.
3. B. Becker, M. Behle, F. Eisenbrand, and R. Wimmer. BDDs in a branch and cut framework. In *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. M. Behle. On Threshold BDDs and the Optimal Variable Ordering Problem. In *COCOA'07: Proceedings of the 1st international conference on Combinatorial optimization and applications*, volume 4616 of *Lecture Notes in Computer Science*, pages 124–135. Springer, 2007.
5. Markus Behle and Friedrich Eisenbrand. 0/1 vertex and facet enumeration with BDDs. In *Proceedings of ALENEX*. SIAM, 2007.
6. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.
7. V. Campos, E. Pi nana, and R. Martí. Adaptive memory programming for matrix bandwidth minimization. *Annals of Operations Research*, To appear.
8. Gianna M. Del Corso and Giovanni Manzini. Finding exact solutions to the bandwidth minimization problem. *Computing*, 62(3):189–203, 1999.
9. Uriel Feige. Approximating the bandwidth via volume respecting embeddings. *J. Comput. Syst. Sci*, 60(3):510–539, 2000.
10. D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pac. J. Math.*, 15:835–855, 1965.
11. Gurari and Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the mincut linear arrangement problem. *ALGORITHMS: Journal of Algorithms*, 5, 1984.
12. 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.
13. T. Hadzic and J. N. Hooker. Cost-bounded binary decision diagrams for 0-1 programming. Technical report, Carnegie Mellon University, 2007.
14. T. Hadzic, J.N. Hooker, B. O'Sullivan, and P. Tiedemann. Approximate Compilation of Constraints into Multivalued Decision Diagrams. In *Proceedings of CP*, volume 5202 of *LNCS*, pages 448–462. Springer, 2008.
15. S. Hoda, W.-J. van Hoeve, and J.N Hooker. A Systematic Approach to MDD-Based Constraint Programming. In *Proceedings of CP*, LNCS. Springer, 2010.
16. J. N. Hooker. *Integrated Methods for Optimization*. Springer, 2007.
17. A.J. Hu. Techniques for Efficient Formal Verification Using Binary Decision Diagrams. Technical Report CS-TR-95-1561, Stanford University, Department of Computer Science, 1995.
18. T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Multi-valued decision diagrams: Theory and applications. *International Journal on Multiple-Valued Logic*, 4:9–62, 1998.
19. C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal*, 38:985–999, 1959.

20. Rafael Martí, Vicente Campos, and Estefanía Piñana. A branch and bound algorithm for the matrix bandwidth minimization. *European Journal of Operational Research*, 186(2):513–528, 2008.
21. Rafael Martí, Manuel Laguna, Fred Glover, and Vicente Campos. Reducing the bandwidth of a sparse matrix with tabu search. *European Journal of Operational Research*, 135(2):450–459, 2001.
22. Estefanía Piñana, Isaac Plana, Vicente Campos, and Rafael Martí. GRASP and path relinking for the matrix bandwidth minimization. *European Journal of Operational Research*, 153(1):200–210, 2004.
23. J. Saxe. Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM J. Algebraic Discrete Meth.*, 1:363–369, 1980.