Discrete Optimization with Decision Diagrams

David Bergman
University of Connecticut

André A. Ciré
University of Toronto

Willem-Jan van Hoeve
Carnegie Mellon University

J. N. Hooker
Carnegie Mellon University

September 2014
To appear in *INFORMS Journal on Computing*

Abstract

We propose a general branch-and-bound algorithm for discrete optimization in which binary decision diagrams (BDDs) play the role of the traditional linear programming relaxation. In particular, relaxed BDD representations of the problem provide bounds and guidance for branching, while restricted BDDs supply a primal heuristic. Each problem is given a dynamic programming model that allows one to exploit recursive structure, even though the problem is not solved by dynamic programming. A novel search scheme branches within relaxed BDDs rather than on values of variables. Preliminary testing shows that a rudimentary BDD-based solver is competitive with or superior to a leading commercial integer programming solver for the maximum stable set problem, the maximum cut problem on a graph, and the maximum 2-satisfiability problem. Specific to the maximum cut problem, we tested the BDD-based solver on a classical benchmark set and identified better solutions and tighter relation bounds than have ever been identified by any technique, nearly closing the entire optimality gap on several large-scale instances.

Keywords. Programming: Integer: Branch-and-bound; Dynamic Programming: Deterministic; Networks/graphs

1 Introduction

Some of the most effective methods for discrete optimization are branch-and-bound algorithms applied to an integer programming formulation of the problem. Linear programming (LP) relaxation plays a central role in these methods, primarily by providing bounds and feasible solutions as well as guidance for branching.

We propose an alternative branch-and-bound method in which *decision diagrams* take over the functions of the traditional LP relaxation. Binary decision diagrams (BDDs) were originally introduced for applications in circuit design and formal verification (Akers 1978, Lee 1959, Bryant 1986, Hu 1995) and have since been used for a variety of other purposes (Wegener 2000, Loekito et al. 2010). A BDD is a graphical representation of a Boolean function that can also be viewed as representing the feasible set of a binary optimization problem. Weights can be associated with arcs of the BDD to represent an objective function, resulting in a *weighted* BDD. Paths in a suitably chosen diagram correspond to feasible solutions of the problem, and a longest (or shortest)

path corresponds to an optimal solution. This raises the possibility of using weighted BDDs as an optimization tool. Although an exact BDD representation of a feasible set tends to explode exponentially, useful relaxations and restrictions of the feasible set can be represented with much smaller BDDs. Furthermore, a relaxed BDD provides the framework for a novel branching scheme. Problems with multi-valued discrete variables are easily accommodated by moving to *multi-valued* decision diagrams, a straightforward extension of BDDs (Kam et al. 1998).

A possible attraction of BDD-based discrete optimization is that it permits an alternative approach to modeling. Modeling for BDDs does not require an inequality formulation, which can be very large, nor does it rely on integer variables or linearization. Rather, the problem is formulated as a dynamic programming (DP) recursion, and memory requirements are controlled by limiting the size of the relaxed and restricted BDDs. This allows one to exploit recursive structure in a given class of problems, much as an LP approach allows one to exploit polyhedral structure. In addition, discrete relaxations based on BDDs allow rapid computation of bounds, as well as an alternative and perhaps more efficient approach to branching.

We therefore propose a general BDD-based optimization method that has the following elements. To fix ideas, assume that we are maximizing.

- A *DP model* of the problem that exploits recursive structure. The size of the DP state space is of little concern, because the problem is not solved by DP or approximate DP. Rather, the state space is chosen to allow construction of an effective BDD relaxation of the problem. This may result in different formulations than are normally used for a DP solution, as we illustrate here.

- A scheme for *merging states*, and perhaps adjusting arc costs, to be used in building a relaxed BDD representation. The relaxation scheme is viewed as part of the DP model, much as valid inequalities can be viewed as part of an integer programming (IP) model. Schemes for merging states are reminiscent of conventional state space relaxation strategies, but they generally differ in three ways: (a) the goal is to provide a tight bound rather than to solve the problem by DP, (b) the relaxation may involve the modification of arc costs, and (c) the merging scheme is dynamic and sensitive to evolving characteristics of the current BDD relaxation as it is built.

- A novel *branch-and-bound* scheme that operates within a BDD relaxation of the problem. Rather than branch on values of a variable, the scheme branches on a suitably chosen subset of nodes in the relaxed BDD. Each node gives rise to a subproblem for which a relaxed BDD can be created, and so on recursively. This sort of branching implicitly enumerates sets of partial solutions, rather than values of one variable. It also takes advantage of information about the search space that is encoded in the structure of the relaxed BDD. The branching nodes are selected on the basis of that structure, rather than on the basis of fractional variables, pseudo-costs, and other information obtained from an LP solution.

- *Upper bounds* obtained from the relaxed BDDs used for branching. To ensure generality of the method, the relaxed BDDs are generated from the DP model using a standardized algorithm. Bounds are obtained by simple longest-path calculations in the relaxed BDDs, rather than from solution of LP relaxations. Memory requirements and the tightness of the bounds are readily adjusted by controlling the size of the BDDs as they are generated.

- *Lower bounds* obtained from restricted BDDs created during the branching procedure. The BDDs are again generated from the DP model in a standard fashion. Shortest-path calculations in restricted BDDs provide a primal heuristic that serves the same function as "feasibility pumps" and other primal heuristics in IP. In addition, a feasible solution is obtained when a BDD relaxation happens to be exact, much as when the solution of an LP relaxation happens to be integral.

The paper is organized as follows. After surveying previous work, we define discrete optimization in general and state three classical optimization problems that will serve as running examples: the maximum independent set problem, the maximum cut problem on a graph, and the maximum 2-satisfiability problem. The problems are selected to illustrate BDD concepts in a variety of settings. They also permit direct computational comparison with an IP solver, because they have straightforward linear inequality formulations.

We then formally define BDDs and show how they can represent feasible solutions and objective function values for discrete optimization problems. We explain how to develop DP models for BDD-based optimization in Sections 5 and 6 and specify how these are used to generate relaxed BDDs. Section 7 shows how to build restricted BDDs, and Section 8 presents the BDD-based branch-and-bound algorithm.

Because our BDD-based solver is proposed as a general-purpose method, it is appropriate to compare it with another general-purpose solver. Integer programming is widely viewed as the most highly developed technology for general discrete optimization, and we therefore compare BDD-based optimization to a leading commercial IP solver in Section 10. We find that although IP solvers have improved by orders of magnitude since their introduction, our rudimentary BDD-based solver is competitive with or superior to the IP state of the art on the problem instances tested here. The paper concludes with a summary and directions for future research.

2 Previous Work

Early applications of BDDs to discrete optimization include cut generation (Becker et al. 2005), 0–1 global optimization (Hooker 2006), post-optimality analysis (Hadžić and Hooker 2006, 2007), and vertex and facet enumeration (Behle and Eisenbrand 2007).

Relaxed BDDs were introduced by Andersen et al. (2007) for the purpose of replacing the domain store used in constraint programming by a richer data structure. Similar methods were applied to other types of constraints in Hadžić et al. (2008a,b), Hoda et al. (2010) and Bergman et al. (2012). Weighted BDD relaxations were used to obtain optimization bounds in Bergman et al. (2011, to appear), the former of which applied them to set covering and the latter to the maximum independent set problem. Restricted BDDs were introduced by Bergman et al. (submitted).

Although we focus here on general-purpose methods, BDD-based optimization can be equally competitive as a problem-specific method. Cire and van Hoeve (to appear) applied weighted BDDs to a variety of sequencing problems, in many cases improving on the state of the art, and in particular closing three sequential ordering problems in TSPLIB. Kell and van Hoeve (2013) developed a special-purpose BDD method for the multidimensional bin packing problem, with results superior to a commercial IP solver, although with no comparison to a special-purpose method.

As noted earlier, BDD-based relaxation is related to state space relaxation in DP (Christofides et al. 1981, Mingozzi 2002, Righini and Salani 2008, Baldacci et al. 2012). State space relaxation approximates the original state space by a smaller, computationally feasible space that allows objective function bounds to be proved or heuristic solutions to be found. As explained above, however, BDD-based relaxation as developed here differs from conventional state space relaxation in several respects.

The relationship between weighted BDDs and DP is further studied in Hooker (2013). In addition, Sanner and McAllester (2005) and Hooker (2013) reallocate transition costs to different arcs of the weighted BDD in a manner that is similar to the DP models proposed here.

3 Discrete Optimization Problems

For our purposes, a *discrete optimization problem* $\mathcal{P}$ has the form $\max\{f(x) \mid x \in D,\ \mathcal{C}\}$, where $x = (x_1, \dots, x_n)$ is a tuple of variables, and $\mathcal{C} = \{C_1, \dots, C_m\}$ is a (possibly empty) constraint set. Each variable x_j has a finite domain D_j, with $D = D_1 \times \cdots \times D_n$. Each constraint C_i is either satisfied or violated by any given x, and $f : D \to \mathbb{R}$ is an objective function. A *feasible solution* of $\mathcal{P}$ is any $x \in D$ that satisfies all of the constraints in $\mathcal{C}$. The set of feasible solutions of $\mathcal{P}$ is denoted by $\text{Sol}(\mathcal{P})$. A feasible solution x^* is *optimal* for $\mathcal{P}$ if it satisfies $f(x^*) \geq f(x)$ for all $x \in \text{Sol}(\mathcal{P})$. Let $z^*(\mathcal{P}) = f(x^*)$ be the optimal value.

To simplify exposition, we restrict our discussion to *binary optimization problems*, in which $|D_j| = 2$ for each j, but the ideas and techniques can be extended to the more general case. We will focus on three particular binary optimization problems as running examples to illustrate the concepts, and on which to conduct computational tests.

3.1 Maximum Independent Set Problem

Given a graph $G = (V, E), V = \{1, 2, \dots, n\}$, an *independent set* I is a subset $I \subseteq V$ such that no two vertices in I are connected by an edge in E. Given weights $w_j \geq 0$ for each vertex $j \in V$, the *maximum independent set problem (MISP)* asks for a maximum-weight independent set of G. The MISP (which is equivalent to the maximum clique problem) has found applications in many areas, including data mining (Edachery et al. 1999), bioinformatics (Eblen et al. 2011), and social network analysis (Balasundaram et al. 2011).

To formulate the MISP as a binary optimization problem, we let variable x_j indicate whether vertex j is selected ($x_j = 1$) or not ($x_j = 0$), for $j \in V$, so that the domain is $D_j = \{0, 1\}$. The objective function is $f(x) = \sum_{j=1}^{n} w_j x_j$, and the constraint set is $\mathcal{C} = \{x_i + x_j \leq 1 \mid (i, j) \in E\}$.

3.2 Maximum Cut Problem

Given a graph $G = (V, E)$, a *cut* (S, T) is a partition of the vertices in V. We say that an edge crosses the cut if its endpoints are on opposite sides of the cut. Given edge weights, the value $v(S, T)$ of a cut is the sum of the weights of the edges crossing the cut. The *maximum cut problem* (MCP) is the problem of finding a cut of maximum value. The MCP has been applied to VLSI design, statistical physics, and other problems (Hager and Krylyuk 1999, Festa et al. 2002).

To formulate the MISP as a binary optimization problem, let x_j indicate the set (S or T) in which vertex j is placed, so that $D_j = \{\text{S}, \text{T}\}$. Using the notation $S(x) = \{j \mid x_j = \text{S}\}$ and

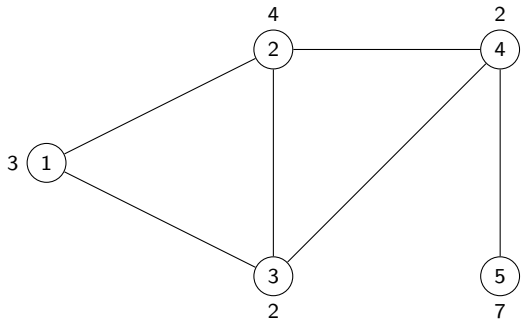


Figure 1: Graph with vertex weights for the MISP.

$T(x) = \{j \mid x_j = \mathrm{T}\}$, the objective function is $f(x) = v(S(x), T(x))$. Since any partition is feasible, $\mathcal{C} = \varnothing$.

3.3 Maximum 2-Satisfiability Problem

Let $x = (x_1, \ldots, x_n)$ be a tuple of Boolean variables, where each x_j can take value T or F (corresponding to true or false). A *literal* is a variable x_j or its negation $\neg\, x_j$. A *clause* c_i is a disjunction of literals, which is satisfied if at least one literal in c_i is true. If $C = \{c_1, \ldots, c_m\}$ is a set of clauses, each with exactly 2 literals, and if each c_i has weight $w_i \geq 0$, the *maximum 2-satisfiability problem* (MAX-2SAT) is the problem of finding an assignment of truth values to $x_1, \ldots, x_n$ that maximizes the sum of the weights of the satisfied clauses in C. MAX-2SAT has applications in scheduling, electronic design automation, computer architecture design, pattern recognition, inference in Bayesian networks, and elsewhere.

To formulate the MAX-2SAT as a binary optimization problem, we use the Boolean variables x_j with domain $D_j = \{\mathrm{F}, \mathrm{T}\}$. The constraint set $\mathcal{C}$ is empty, and the objective function is $f(x) = \sum_{i=1}^{m} w_i c_i(x)$, where $c_i(x) = 1$ if x satisfies clause c_i, and $c_i(x) = 0$ otherwise.

4 Binary Decision Diagrams

4.1 Concepts and Notation

A *binary decision diagram* (BDD) $B = (U, A, d)$ is a layered directed acyclic multi-graph (U, A) with labeled arcs that encode values of binary variables. The node set U is partitioned into layers $L_1, L_2, \ldots, L_{n+1}$, where layers L_1 and L_{n+1} consist of single nodes, the root r and the terminal t, respectively. Each arc $a \in A$ is directed from a node in some L_j to a node in L_{j+1} and has a label $d(a) \in \{0, 1\}$ that represents the value of a binary variable x_j. No two arcs leaving the same node have the same label, which means every node has a maximum out-degree of 2.

Figure 2 presents examples of BDDs, where the dashed arcs are 0-arcs (i.e., arcs with label 0), and the solid arcs are 1-arcs. We let $a_0(u)$ denote the 0-arc leaving node u (if it exists), and similarly for $a_1(u)$. We also let $b_0(u)$ denote the node at the opposite end of arc $a_0(u)$, and similarly for $b_1(u)$. Every arc-specified path $p = (a_1, \ldots, a_n)$ from r to t encodes an assignment to the binary variables $x_1, \ldots, x_n$, namely $x_j = d(a_j)$ for $j = 1, \ldots, n$. We will denote this assignment by x^p. The set of r–t paths represents a set of assignments we denote $\mathrm{Sol}(B)$.

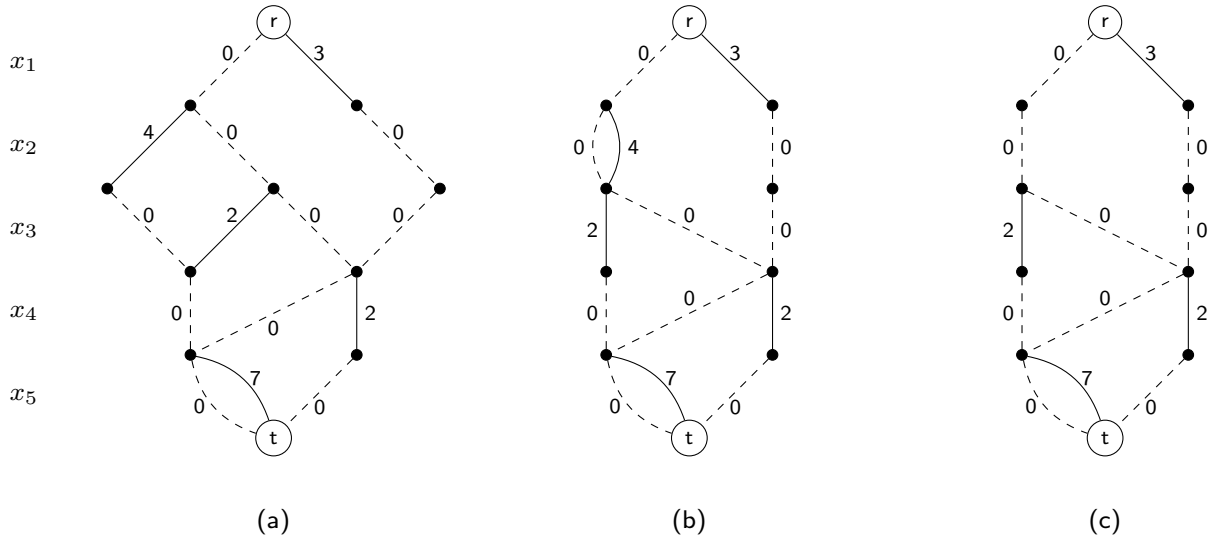


Figure 2: (a) Exact BDD, (b) relaxed BDD, and (c) restricted BDD for the MISP on the graph in Figure 1.

The *width* $|L_j|$ of layer L_j is the number of nodes in the layer, and the *width* of a BDD B is $\max_j\{|L_j|\}$. The *size* $|B|$ of B is the number of nodes in B.

It is common in the BDD literature to allow various types of *long arcs* that skip one or more layers (Bryant 1986, Minato 1993). Long arcs can improve efficiency because they represent multiple partial assignments with a single arc, but to simplify exposition, we will suppose with minimal loss of generality that there are no long arcs. BDDs also typically have two terminal nodes, corresponding to true and false, but for our purposes only a true node is required—as the terminus for feasible paths.

A *multi-valued decision diagram* allows out-degrees higher than 2 and therefore encodes values of general finite-domain variables. All the ideas presented here, including the branch-and-bound algorithm of Section 8, are easily extended to allow for MDDs and general finite domains.

Because we are interested in optimization, we focus on *weighted* BDDs, in which each arc a has an associated *length* $v(a)$. The length of a directed path $p = (a_1, \ldots, a_k)$ rooted at r is its length $v(p) = \sum_{j=1}^{k} v(a_j)$.

A weighted BDD B represents a binary optimization problem $\mathcal{P}$ in a straightforward way. It can be an exact representation, a relaxation, or a restriction. B is an exact representation of $\mathcal{P}$ if the r–t paths in B encode precisely the feasible solutions of $\mathcal{P}$, and the length of a path is the objective function value of the corresponding solution. More formally, we say that B is *exact* for $\mathcal{P}$ when

$$\text{Sol}(\mathcal{P}) \;=\; \text{Sol}(B) \tag{E-1}$$

$$f(x^p) \;=\; v(p), \text{ for all } r\text{–}t \text{ paths } p \text{ in } B \tag{E-2}$$

B is *relaxed* for $\mathcal{P}$ if B represents a superset of the feasible solutions of $\mathcal{P}$, and path lengths are

upper bounds on the value of feasible solutions. That is, B is relaxed for $\mathcal{P}$ if

$$\text{Sol}(\mathcal{P}) \;\subseteq\; \text{Sol}(B) \tag{Rel-1}$$

$$f(x^p) \;\leq\; v(p), \text{ for all } r\text{–}t \text{ paths } p \text{ in } B \text{ for which } x^p \in \text{Sol}(\mathcal{P}) \tag{Rel-2}$$

Finally, B is *restricted* for $\mathcal{P}$ if it represents a subset of the feasible solutions of $\mathcal{P}$, and path lengths are lower bounds on the objective function value. So B is restricted for $\mathcal{P}$ if

$$\text{Sol}(\mathcal{P}) \;\supseteq\; \text{Sol}(B) \tag{Res-1}$$

$$f(x^p) \;\geq\; v(p), \text{ for all } r\text{–}t \text{ paths } p \text{ in } B \tag{Res-2}$$

An exact BDD reduces discrete optimization to a longest-path problem. If p is a longest path in a BDD B that is exact for $\mathcal{P}$, then x^p is an optimal solution of $\mathcal{P}$, and its length $v(p)$ is the optimal value $z^*(\mathcal{P}) = f(x^p)$ of $\mathcal{P}$. When B is relaxed for $\mathcal{P}$, a longest path p provides an upper bound on the optimal value. The corresponding solution x^p may not be feasible, but $v(p) \geq z^*(\mathcal{P})$. Conversely, a longest path calculation serves as a primal heuristic when B is restricted for $\mathcal{P}$. In this case, a longest path p corresponds to a feasible solution x^p of $\mathcal{P}$ that yields a lower bound $v(p)$ on $z^*(\mathcal{P})$.

We now illustrate exact, relaxed, and restricted BDDs for the three problems introduced earlier.

4.2 Example BDDs for the MISP

Consider the graph and vertex weights depicted in Figure 1. Figure 2(a) represents an exact BDD in which each path corresponds to an independent set encoded by the arc labels along the path, and each independent set corresponds to some path. A 1-arc leaving layer L_j indicates that vertex j is in the independent set, and a 0-arc indicates that it is not. The longest r–t path in the BDD has value 11, corresponding to solution $x = (0, 1, 0, 0, 1)$ and to the independent set $\{2, 5\}$, the maximum-weight independent set in the graph.

Figure 2(b) shows a relaxed BDD. Each independent set corresponds to a path, but there are paths p for which x^p is infeasible (i.e., not an independent set). For example, the path $\bar{p}$ encoding $x^{\bar{p}} = (0, 1, 1, 0, 1)$ does not represent an independent set because both endpoints of edge $(2, 3)$ are selected. The length of each path that represents an independent set is the weight of that set, making this a relaxed BDD. The longest path in the BDD is $\bar{p}$, proving an upper bound of 13.

Figure 2(c) represents a restricted BDD. For each path p, x^p is feasible and has length equal to the weight of the corresponding independent set. The longest path corresponds to solution $(1, 0, 0, 0, 1)$ and independent set $\{1, 5\}$, and thus proves a lower bound of 10 on the objective function.

4.3 Example BDDs for MCP

Consider the graph and edge weights in Figure 3. Figure 4(a) depicts an exact BDD for the MCP on this graph. A 0-arc leaving L_j indicates that $x_j = \text{S}$, and a 1-arc indicates $x_j = \text{T}$. Note that we can place vertex 1 in set S without loss of generality. For now, assume the arc lengths are as shown; we derive these formally in Section 5.2. The longest path p corresponds to the optimal solution $x^p = (\text{S}, \text{S}, \text{T}, \text{S})$, and its length 4 is the weight of the maximum cut $(S, T) = (\{1, 2, 4\}, \{3\})$.

Figure 4 (b) depicts a relaxed BDD. Again, every possible partition is represented by some path, and the length of each r–t path is greater than or equal to the weight of the cut. The longest

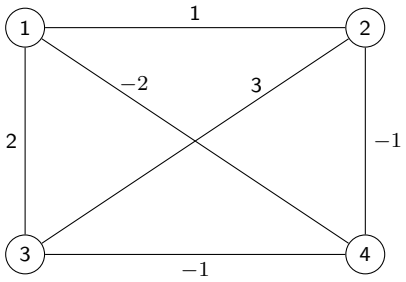


Figure 3: Graph with edge weights for the MCP .

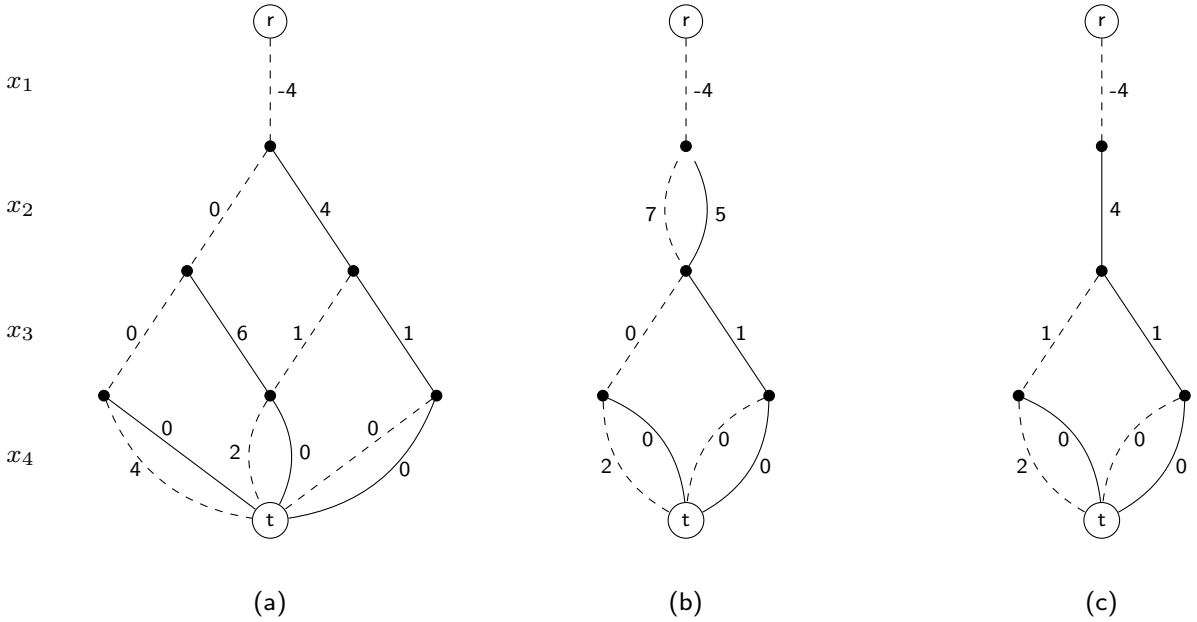


Figure 4: (a) Exact BDD, (b) relaxed BDD, and (c) restricted BDD for the MCP on the graph in Figure 3.

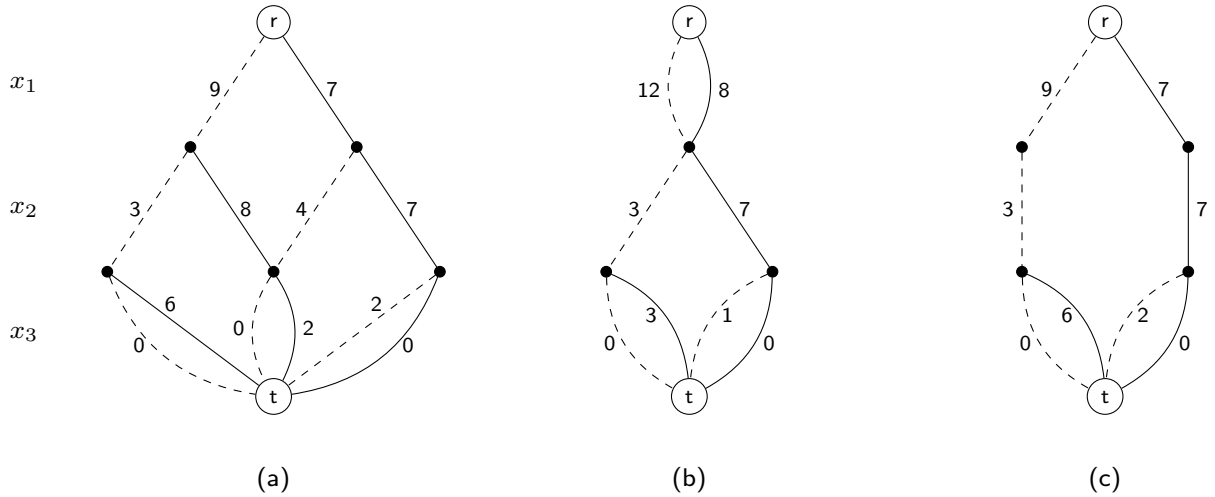


Figure 5: (a) Exact BDD, (b) relaxed BDD, and (c) restricted BDD for the MAX-2SAT instance in Section 4.4.

path corresponds to the solution $(\mathrm{S}, \mathrm{S}, \mathrm{S}, \mathrm{S})$ and has length 5, while the actual weight of this cut is 0. This proves an upper bound of 5 on the objective function.

Figure 4 (c) depicts a restricted BDD. Although not all of the solutions are present in the BDD, each solution that is represented by a path has weight equal to the path length. The longest path corresponds to solution $x = (\mathrm{S}, \mathrm{T}, \mathrm{S}, \mathrm{S})$ and has length 3, a lower bound on the optimal value.

4.4 Example BDDs for MAX-2SAT

Consider the following instance of MAX-2SAT:

clause index	clause	weight
1	$x_1 \vee x_3$	3
2	$\neg x_1 \vee \neg x_3$	5
3	$\neg x_1 \vee x_3$	4
4	$x_2 \vee \neg x_3$	2
5	$\neg x_2 \vee \neg x_3$	1
6	$x_2 \vee x_3$	5

Figure 5 (a) depicts an exact BDD in which 0-arcs leaving L_j assign F to x_j and 1-arcs assign T. There is a path for each possible truth assignment to the variables, and its length is the sum of the weights of the clauses satisfied by that solution. The arc lengths will be derived formally in Section 5.3. The longest path corresponds to $x = (\mathrm{F}, \mathrm{T}, \mathrm{T})$ and has length 19 because it satisfies all clauses but c_5.

Figure 5 (b) shows a relaxed BDD, in which path lengths over-approximate objective function values. The longest path corresponds to $x = (\mathrm{F}, \mathrm{T}, \mathrm{F})$ and has length 20, an upper-bound on

the optimal value. A restricted BDD appears in Figure 5 (c). Each path length is equal to the objective function value of the corresponding solution. The longest path corresponds to solution $x = (\mathrm{F}, \mathrm{F}, \mathrm{T})$ and has length 18, a lower bound on the optimal value.

5 Problem Formulation: The DP Model

We now address the issue of formulating a discrete optimization problem $\mathcal{P}$ for BDD-based solution. The formulation consists of a dynamic programming (DP) model and a rule for merging nodes that are associated with similar states. The DP model serves as the conceptual basis for an exact BDD, which is too large for practical use. The merger rule allows us to build a relaxed BDD of any desired maximum width.

A DP model for a given problem $\mathcal{P}$ consists of the following elements:

- state spaces $S_1, \dots, S_{n+1}$ with $S_1 = \{\hat{r}\}$ and $S_{n+1} = \{\hat{t}, \hat{0}\}$, where $\hat{r}$ is the *root state*, $\hat{t}$ is the *terminal state*, $\hat{0}$ is the *infeasible state*, and $\hat{0} \in S_j$ for $j = 2, \dots, n+1$
- transition functions $t_j : S_j \times D_j \to S_{j+1}$ for $j = 1, \dots, n$, where $t_j(\hat{0}, d) = \hat{0}$ for any $d \in D_j$
- transition cost functions $h_j : S \times D_j \to \mathbb{R}$ for $j = 1, \dots, n$
- a root value v_r that will be added to the arcs directed out of the root node of the BDD

The problem variables $x_1, \dots, x_n$ are regarded as controls, where a given control x_j takes the system from a given state $s^j \in S_j$ to state $t_j(s^j, x_j)$ and incurs cost $h_j(s^j, x_j)$. The DP formulation has variables $(s, x) = (s^1, \dots, s^{n+1}, x_1, \dots, x_n)$ and is written

$$\begin{aligned} &\min\ \hat{f}(s,x) = v_r + \sum_{i=1}^{n} h_j(s^j, x_j) \\ &\text{subject to} \\ &s^{j+1} = t_j(s^j, x_j),\ x_j \in D_j,\ j = 1, \dots, n \\ &s^j \in S_j,\ j = 1, \dots, n+1 \end{aligned} \tag{DP}$$

Let $S = S_1 \times \cdots \times S_{n+1}$. The formulation (DP) is *valid* for $\mathcal{P}$ if for every $x \in D$, there is an $s \in S$ such that (s, x) is feasible in (DP) and

$$s^{n+1} = \hat{t} \text{ and } \hat{f}(s,x) = f(x),\ \text{if } x \text{ is feasible for } \mathcal{P} \tag{A1}$$

$$s^{n+1} = \hat{0},\ \text{if } x \text{ is infeasible for } \mathcal{P} \tag{A2}$$

A valid DP formulation leads directly to an exact BDD representation. Suppose that (DP) is valid for problem $\mathcal{P}$, and consider the state-transition graph for (DP). Omit all occurrences of the infeasible state $\hat{0}$, and let each remaining arc from state s^j to state $t_j(s^j, x_j)$ have length equal to the transition cost $h_j(s^j, x_j)$. The resulting multigraph B_{DP} is an exact BDD for $\mathcal{P}$, because paths from state r to state t in B_{DP} correspond precisely to feasible solutions of (DP), and the objective function value of the corresponding solution is the path length.

The construction of B_{DP} is straightforward in principle. Begin with the root note r in layer 1, which corresponds to the root state $\hat{r}$. Proceed recursively, creating a node for each feasible state

Algorithm 1 Exact or relaxed BDD Compilation

1: Create node $r = \hat{r}$ and let $L_1 = \{r\}$
2: **for** $j = 1$ to n **do**
3: **while** $|L_j| > W$ **do**
4: let $M = \text{node\_select}(L_j)$, $L_j \leftarrow (L_j \setminus M) \cup \{\oplus(M)\}$
5: **for all** $u \in L_{j-1}$ and $i \in \{d_1, d_2\}$ with $b_i(u) \in M$ **do**
6: $b_i(u) \leftarrow \oplus(M)$, $v\left(a_i(u)\right) \leftarrow \Gamma_M\left(v(a_i(u)), b_i(u)\right)$
7: let $L_{j+1} = \varnothing$
8: **for all** $u \in L_j$ and $i \in \{d_1, d_2\}$ **do**
9: let $u' = t_j(u, i)$, add u' to L_{j+1}, and set $b_i(u) = u'$

that can be reached from r. Thus, having constructed layer j, let L_{j+1} contain nodes corresponding to all *distinct* feasible states to which one can transition from states represented in L_j. Then add an arc from layer j to layer $j+1$ for each such transition.

The procedure is more precisely stated as Algorithm 1. Because distinct nodes always have distinct states, the algorithm identifies each node with the state associated with that node. When creating an exact BDD, we let $W = \infty$, so that lines 3–6 of the algorithm have no effect and can be ignored for now.

We do not actually build an exact BDD, because the state-transition graph can grow exponentially with the problem size. Rather, we construct relaxed and restricted BDDs whose size can be controlled by bounding their width. A relaxed BDD can be obtained by selecting a maximum width W and using suitable operators $\oplus$, Γ in Algorithm 1 as discussed in Section 6.

We remark in passing that the resulting BDD is not necessarily *reduced* (Bryant 1986, Wegener 2000), meaning that a layer j may contain two or more nonequivalent nodes. Two nodes are equivalent when the paths from each to t correspond to the same set of assignments to $(x_j, \ldots, x_n)$. In a reduced BDD, all equivalent nodes in a layer are superimposed. Although reduced BDDs play a key role in circuit verification and some other applications, they can be unsuitable for optimization, because the arc lengths from equivalent nodes may differ (Hooker 2013).

We next state DP models for the three discrete optimization problems discussed earlier.

5.1 Formulating the MISP: DP Model

We present a valid DP formulation for the MISP (Bergman et al. 2012, to appear). Let $G = (V, E)$ be a vertex-weighted graph with vertex weight w_i for all $i \in V = \{1, \ldots, n\}$. Define $V_j = \{j, j+1, \ldots, n\}$, and let $N(j) = \{j' \mid (j, j') \in E\} \cup \{j\}$ be the neighborhood of j.

A state $s^j \in S_j$ is the set of vertices that can still be added to obtain an independent set, given the vertices already selected. The transition cost from s^j is w_j if vertex j is selected, and otherwise zero. Formally, the DP model is

- state spaces: $S_j = 2^{V_j}$ for $j = 2, \ldots, n$, $\hat{r} = V$, and $\hat{t} = \varnothing$
- transition functions: $t_j(s^j, 0) = s^j \setminus \{j\}$, $\quad t_j(s^j, 1) = \begin{cases} s^j \setminus N(j) & , \text{ if } j \in s^j \\ \hat{0} & , \text{ if } j \notin s^j \end{cases}$
- cost functions: $h_j(s^j, 0) = 0$, $h_j(s^j, 1) = w_j$

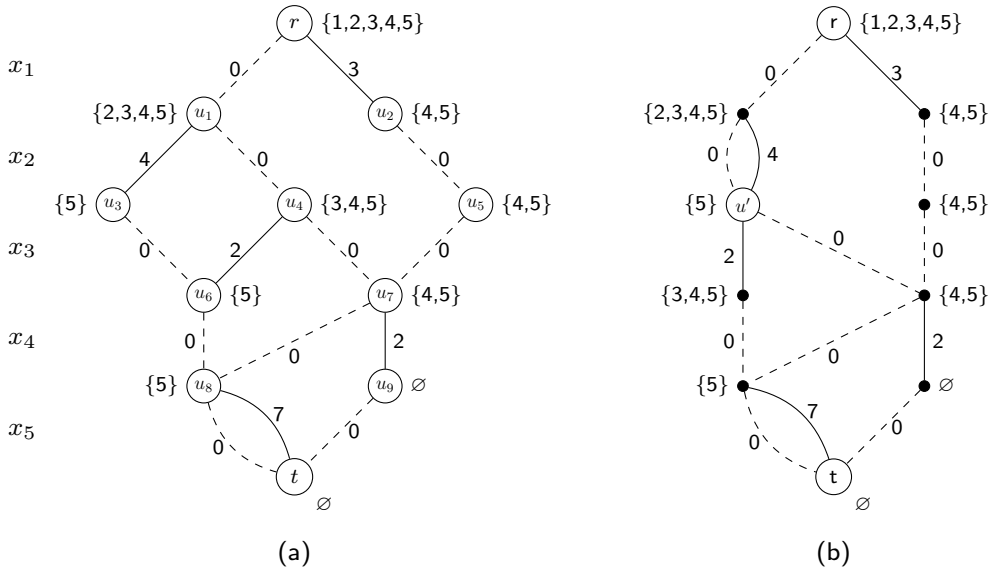


Figure 6: (a) Exact BDD with states for the MISP on the graph in Figure 1. (b) Relaxed BDD for the same problem instance.

- root value: $v_r = 0$

As an illustration, consider the MISP for the graph in Figure 1. The states associated with nodes of B_{DP} are shown in Figure 6(a). For example, node u_1 has state $\{2, 3, 4, 5\}$, representing the vertex set $V \setminus \{1\}$. The following is proved in (Bergman et al. to appear).

Theorem 1 *The above specifications yield a valid DP formulation of the MISP.*

5.2 Formulating the MCP: DP Model

We now formulate a DP model for the MPC. Let $G = (V, E)$ be an edge-weighted graph, which we can assume (without loss of generality) to be complete, because missing edges can be included with weight 0. A natural state variable s^j would be the set of vertices already placed in S, as this is sufficient to determine the transition cost of the next choice. However, we will be interested in merging nodes that lead to similar objective function values. We therefore let the state indicate, for vertex $j, \ldots, n$, the net marginal benefit of placing that vertex in T, given previous choices. We will show that this is sufficient information to construct a DP recursion.

Formally, we specify the DP formulation as follows. As before, the control variable is $x_j \in \{\text{S}, \text{T}\}$, indicating in which set vertex j is placed, and we set $x_1 = \text{S}$ without loss of generality. We will use the notation $(\alpha)^+ = \max\{\alpha, 0\}$ and $(\alpha)^- = \min\{\alpha, 0\}$.

- state spaces: $S_k = \left\{ s^k \in \mathbb{R}^n \mid s^k_j = 0,\ j = 1, \ldots, k-1 \right\}$, with root state and terminal state equal to $(0, \ldots, 0)$

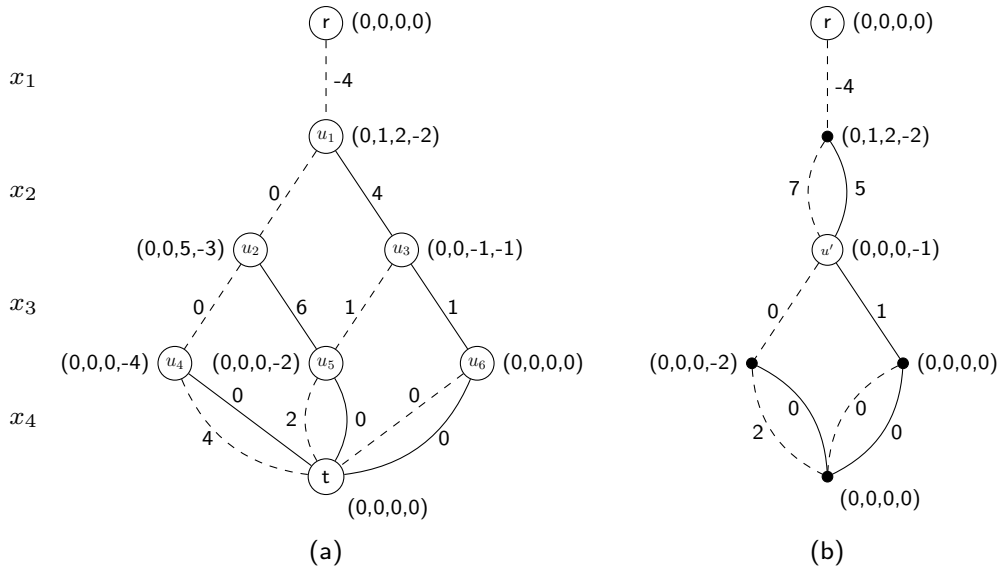


Figure 7: (a) Exact BDD with states for the MCP on the graph in Figure 3. (b) Relaxed BDD for the same problem instance.

- transition functions: $t_k(s^k, x_k) = (0, \ldots, 0, s^{k+1}_{k+1}, \ldots, s^{k+1}_n)$, where

$$s^{k+1}_\ell = \left\{ \begin{array}{ll} s^k_\ell + w_{k\ell}, & \text{if } x_k = \text{S} \\ s^k_\ell - w_{k\ell}, & \text{if } x_k = \text{T} \end{array} \right\}, \quad \ell = k+1, \ldots, n$$

- transition cost: $h_1(s^1, x_1) = 0$ for $x_1 \in \{\text{S}, \text{T}\}$, and

$$h_k(s^k, x_k) = \left\{ \begin{array}{ll} (-s_k)^+ + \displaystyle\sum_{\substack{\ell > k \\ s^j_\ell w_{j\ell} \le 0}} \min\left\{|s^j_\ell|, |w_{j\ell}|\right\}, & \text{if } x_k = \text{S} \\ (s_k)^+ + \displaystyle\sum_{\substack{\ell > k \\ s^j_\ell w_{j\ell} \ge 0}} \min\left\{|s^j_\ell|, |w_{j\ell}|\right\}, & \text{if } x_k = \text{T} \end{array} \right\}, \quad k = 2, \ldots, n$$

- root value: $v_r = \displaystyle\sum_{1 \le j < j' \le n} (w_{jj'})^-$

Note that the root value is the sum of the negative arc weights. The state transition is based on the fact that if vertex k is added to S, then the marginal benefit of placing vertex $\ell > k$ in T (given choices already made for vertices $1, \ldots, k-1$) is increased by $w_{k\ell}$. If k is added to T, the marginal benefit is reduced by $w_{k\ell}$. Figure 7(a) shows the resulting weighted BDD for the example discussed earlier. We now prove that the transition costs correctly capture the objective function.

Theorem 2 *The above is a valid DP formulation of the MCP.*

Proof. Note that any solution $x \in \{S,T\}^n$ is feasible so that we need only show (A1). The state transitions clearly imply that s^{n+1} is the terminal state $\hat{t} = (0,\ldots,0)$, and thus $s^{n+1} \in \{\hat{t},\hat{0}\}$. If we let $(\bar{s},\bar{x})$ be an arbitrary solution of (DP), it remains to show that $\hat{f}(\bar{s},\bar{x}) = f(\bar{x})$. Let H_k be the sum of the first k transition costs for solution $(\bar{s},\bar{x})$, so that $H_k = \sum_{j=1}^{k} h_j(\bar{s}^j,\bar{x}_j)$ and $H_n + v_r = \hat{f}(\bar{s},\bar{x})$. It suffices to show that

$$H_n + v_r = \sum_{j,j'} \left\{ w_{jj'} \mid 1 \le j < j' \le n,\ \bar{x}_j \ne \bar{x}_{j'} \right\} \tag{Hn}$$

because the right-hand side is $f(\bar{x})$. We prove (Hn) as follows. Note first that the state transitions imply that

$$s^k_\ell = L^\ell_{k-1} - R^\ell_{k-1}, \ \text{ for } \ell \ge k \tag{Sk}$$

where

$$L^\ell_k = \sum_{\substack{j \le k \\ \bar{x}_j = \mathrm{S}}} x_{j\ell}, \quad R^\ell_k = \sum_{\substack{j \le k \\ \bar{x}_j = \mathrm{T}}} x_{j\ell}, \ \text{ for } \ell > k$$

We will show the following inductively:

$$H_k + N_k = \sum_{\substack{j<j' \le k \\ \bar{x}_j \ne \bar{x}_{j'}}} w_{jj'} + \sum_{\ell > k} \min\left\{ L^\ell_k, R^\ell_k \right\} \tag{Hk}$$

where N_k is a partial sum of negative arc weights, specifically

$$N_k = \sum_{j<j' \le k} (w_{jj'})^- + \sum_{j \le k < \ell} (w_{j\ell})^-$$

so that, in particular, $N_n = v_r$. This proves the theorem, because (Hk) implies (Hn) when $k = n$.

We first note that (Hk) holds for $k = 1$, because in this case both sides vanish. We now suppose (Hk) holds for $k-1$ and show that it holds for k. The definition of transition cost implies

$$H_k = H_{k-1} + (\sigma_k s^k_k)^+ + \sum_{\substack{\ell > k \\ \sigma_k s^k_\ell w_{k\ell} \ge 0}} \min\left\{ |s^k_\ell|, |w_{k\ell}| \right\}$$

where σ_k is 1 if $\bar{x}_k = \mathrm{T}$ and -1 otherwise. This and the inductive hypothesis imply

$$H_k = \sum_{\substack{j<j' \le k-1 \\ \bar{x}_j \ne \bar{x}_{j'}}} w_{jj'} + \sum_{\ell \ge k} \min\left\{ L^\ell_{k-1}, R^\ell_{k-1} \right\} - N_{k-1} + (\sigma_k s^k_k)^+ + \sum_{\substack{\ell > k \\ \sigma_k s^k_\ell w_{k\ell} \ge 0}} \min\left\{ |s^k_\ell|, |w_{k\ell}| \right\}$$

We wish to show that this is equal to the right-hand side of (Hk) minus N_k. Making the substitution (Sk) for state variables, we can establish this equality by showing

$$\begin{aligned} &\sum_{\ell \ge k} \min\left\{ L^\ell_{k-1}, R^\ell_{k-1} \right\} - N_{k-1} + (\sigma_k (L^k_{k-1} - R^k_{k-1}))^+ + \sum_{\substack{\ell > k \\ \sigma_k (L^\ell_{k-1} - R^\ell_{k-1}) w_{k\ell} \ge 0}} \min\left\{ |L^\ell_{k-1} - R^\ell_{k-1}|, |w_{k\ell}| \right\} \\ &\quad = \sum_{\substack{j<k \\ \bar{x}_j \ne \bar{x}_k}} w_{jk} + \sum_{\ell > k} \min\left\{ L^\ell_k, R^\ell_k \right\} - N_k \end{aligned} \tag{Eq1}$$

We will show that (Eq1) holds when $\bar{x}_k = \mathrm{T}$. The proof for $\bar{x}_k = \mathrm{S}$ is analogous. Using the fact that $R^\ell_k = R^\ell_{k-1} + w_{k\ell}$, (Eq1) can be written

$$\begin{aligned}\min\left\{L^k_{k-1}, R^k_{k-1}\right\} + \sum_{\ell>k} \min\left\{L^\ell_{k-1}, R^\ell_{k-1}\right\} + (L^k_{k-1} - R^k_{k-1})^+ \\ + \sum_{\substack{\ell>k \\ (L^\ell_{k-1}-R^\ell_{k-1})w_{k\ell}\geq 0}} \min\left\{|L^\ell_{k-1} - R^\ell_{k-1}|, |w_{k\ell}|\right\} \\ = L^k_{k-1} + \sum_{\ell>k} \min\left\{L^\ell_{k-1}, R^\ell_{k-1} + w_{k\ell}\right\} - (N_k - N_{k-1})\end{aligned} \tag{Eq2}$$

The first and third terms of the left-hand side of (Eq2) sum to L^k_{k-1}. We can therefore establish (Eq2) by showing that for each $\ell \in \{k+1, \ldots, n\}$, we have

$$\begin{aligned}\min\left\{L^\ell_{k-1}, R^\ell_{k-1}\right\} + \delta \min\left\{R^\ell_{k-1} - L^\ell_{k-1}, -w_{k\ell}\right\} &= \min\left\{L^\ell_{k-1}, R^\ell_{k-1} + w_{k\ell}\right\} - w_{k\ell}, \quad \text{if } w_{k\ell} < 0 \\ \min\left\{L^\ell_{k-1}, R^\ell_{k-1}\right\} + (1-\delta) \min\left\{L^\ell_{k-1} - R^\ell_{k-1}, w_{k\ell}\right\} &= \min\left\{L^\ell_{k-1}, R^\ell_{k-1} + w_{k\ell}\right\}, \quad \text{if } w_{k\ell} \geq 0\end{aligned}$$

where $\delta = 1$ if $L^\ell_{k-1} \leq R^\ell_{k-1}$ and $\delta = 0$ otherwise. It is easily checked that both equations are identities. □

5.3 Formulating MAX-2SAT: DP Model

We suppose without loss of generality that a MAX-2SAT problem contains all $4 \cdot \binom{n}{2}$ possible clauses, because missing clauses can be given zero weight. Thus $\mathcal{C}$ contains $x_j \vee x_k$, $x_j \vee \neg x_k$, $\neg x_j \vee x_k$ and $\neg x_j \vee \neg x_k$ for each pair $j, k \in \{1, \ldots, n\}$ with $j \neq k$. Let w^{TT}_{jk} be the weight assigned to $x_j \vee x_k$, w^{TF}_{jk} the weight assigned to $x_j \vee \neg x_k$, and so forth.

We let each state variable s^k be an array $(s^k_1, \ldots, s^k_n)$ in which each s^k_j is the net benefit of setting x_j to true, given previous settings. The net benefit is the advantage of setting $x_j = \mathrm{T}$ over setting $x_j = \mathrm{F}$. Suppose, for example, that $n = 2$ and we have fixed $x_1 = \mathrm{T}$. Then $x_1 \vee x_2$ and $x_1 \vee \neg x_2$ are already satisfied. The value of x_2 makes no difference for them, but setting $x_2 = \mathrm{T}$ newly satisfies $\neg x_1 \vee x_2$, while $x_2 = \mathrm{F}$ newly satisfies $\neg x_1 \vee \neg x_2$. Setting $x_2 = \mathrm{T}$ therefore obtains net benefit $w^{\mathrm{FT}}_{12} - w^{\mathrm{FF}}_{12}$. If x_1 has not yet been assigned a truth value, then we do not compute a net benefit for setting $x_2 = \mathrm{T}$. Formally, the DP formulation is as follows.

- state spaces: $S_k = \left\{s^k \in \mathbb{R}^n \mid s^k_j = 0,\ j = 1, \ldots, k-1\right\}$, with root state and terminal state equal to $(0, \ldots, 0)$
- transition functions: $t_k(s^k, x_k) = (0, \ldots, 0, s^{k+1}_{k+1}, \ldots, s^{k+1}_n)$, where

$$s^{k+1}_\ell = \left\{ \begin{array}{ll} s^k_\ell + w^{\mathrm{TT}}_{k\ell} - w^{\mathrm{TF}}_{k\ell}, & \text{if } x_k = \mathrm{F} \\ s^k_\ell + w^{\mathrm{FT}}_{k\ell} - w^{\mathrm{FF}}_{k\ell}, & \text{if } x_k = \mathrm{T} \end{array} \right\}, \quad \ell = k+1, \ldots, n$$

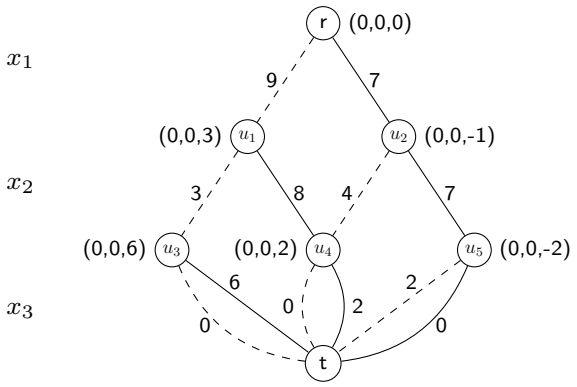


Figure 8: Exact BDD with states for the MAX-2SAT problem introduced in Section 4.4.

- transition cost: $h_1(s^1, x_1) = 0$ for $x_1 \in \{\mathrm{F}, \mathrm{T}\}$, and

$$h_k(s^k, x_k) = \left\{ \begin{array}{ll} (-s_k^k)^+ + \sum_{\ell>k} \left(w_{k\ell}^{\mathrm{FF}} + w_{k\ell}^{\mathrm{FT}} + \min\left\{ (s_\ell^k)^+ + w_{k\ell}^{\mathrm{TT}}, (-s_\ell^k)^+ + w_{k\ell}^{\mathrm{TF}} \right\} \right), & \text{if } x_k = \mathrm{F} \\ (s_k^k)^+ + \sum_{\ell>k} \left(w_{k\ell}^{\mathrm{TF}} + w_{k\ell}^{\mathrm{TT}} + \min\left\{ (s_\ell^k)^+ + w_{k\ell}^{\mathrm{FT}}, (-s_\ell^k)^+ + w_{k\ell}^{\mathrm{FF}} \right\} \right), & \text{if } x_k = \mathrm{T} \end{array} \right\},$$
$$k = 2, \ldots, n$$

- root value: $v_r = 0$

Figure 8(a) shows the resulting states and transition costs for the example discussed earlier.

Theorem 3 *The above is a valid DP formulation of the MAX-2SAT problem.*

Proof. Since any solution $x \in \{F, T\}^n$ is feasible, we need only show that the costs are correctly computed. Thus if $(\bar{s}, \bar{x})$ is an arbitrary solution of (DP), we wish to show that $\hat{f}(\bar{s}, \bar{x}) = f(\bar{x})$. If H_k is as before, we wish to show that $H_n = \mathrm{SAT}_n(\bar{x})$, where $\mathrm{SAT}_k(\bar{x})$ is the total weight of clauses satisfied by the settings $\bar{x}_1, \ldots, \bar{x}_k$. Thus

$$\mathrm{SAT}_k(\bar{x}) = \sum_{jj'\alpha\beta} \{ w_{jj'}^{\alpha\beta} \mid 1 \le j < j' \le k;\ \alpha, \beta \in \{\mathrm{F}, \mathrm{T}\};\ \bar{x}_j = \alpha \text{ or } \bar{x}_{j'} = \beta \}$$

Note first that the state transitions imply (Sk) as in the previous proof, where

$$L_k^\ell = \sum_{\substack{1 \le j \le k \\ \bar{x}_j = \mathrm{T}}} w_{j\ell}^{\mathrm{FT}} + \sum_{\substack{1 \le j \le k \\ \bar{x}_j = \mathrm{F}}} w_{j\ell}^{\mathrm{TT}}, \quad R_k^\ell = \sum_{\substack{1 \le j \le k \\ \bar{x}_j = \mathrm{T}}} w_{j\ell}^{\mathrm{FF}} + \sum_{\substack{1 \le j \le k \\ \bar{x}_j = \mathrm{F}}} w_{j\ell}^{\mathrm{TF}}, \ \text{ for } \ell > k$$

We will show the following inductively:

$$H_k = \mathrm{SAT}_k(\bar{x}) + \sum_{\ell>k} \min\left\{ L_k^\ell, R_k^\ell \right\} \qquad \text{(Hk-SAT)}$$

This proves the theorem, because (Hk-SAT) reduces to $H_n = \mathrm{SAT}_n(\bar{x})$ when $k = n$.

To simplify the argument, we begin the induction with $k = 0$, for which both sides of (Hk-SAT) vanish. We now suppose (Hk-SAT) holds for $k-1$ and show that it holds for k. The definition of transition cost implies

$$H_k = H_{k-1} + (\sigma_k s_k^k)^+ + \sum_{\ell>k} \left(w_{k\ell}^{\alpha \mathrm{F}} + w_{k\ell}^{\alpha \mathrm{T}} + \min \left\{ (s_\ell^k)^+ + w_{k\ell}^{\beta \mathrm{T}}, (-s_\ell^k)^+ + w_{k\ell}^{\beta \mathrm{F}} \right\} \right)$$

where σ_k is 1 if $\bar{x}_k = \mathrm{T}$ and -1 otherwise. Also α is the truth value $\bar{x}_k$ and β is the value opposite $\bar{x}_k$. This and the inductive hypothesis imply

$$H_k = \mathrm{SAT}_{k-1}(\bar{x}) + \sum_{\ell \geq k} \min \left\{ L_k^\ell, R_k^\ell \right\} + (\sigma_k s_k^k)^+ + \sum_{\ell>k} \left(w_{k\ell}^{\alpha \mathrm{F}} + w_{k\ell}^{\alpha \mathrm{T}} + \min \left\{ (s_\ell^k)^+ + w_{k\ell}^{\beta \mathrm{T}}, (-s_\ell^k)^+ + w_{k\ell}^{\beta \mathrm{F}} \right\} \right)$$

We wish to show that this is equal to the right-hand side of (Hk-SAT). We will establish this equality on the assumption that $\bar{x}_k = \mathrm{T}$, as the proof is analogous when $\bar{x}_k = \mathrm{F}$. Making the substitution (Sk) for state variables, and using the facts that $L_k^\ell = L_{k-1}^\ell + w_{k\ell}^{\mathrm{FT}}$ and $R_k^\ell = R_{k-1}^\ell + w_{k\ell}^{\mathrm{FF}}$, it suffices to show

$$\begin{aligned} &\sum_{\ell>k} \min \left\{ L_k^\ell, R_k^\ell \right\} + \min \left\{ L_{k-1}^k, R_{k-1}^k \right\} + (L_{k-1}^k - R_{k-1}^k)^+ \\ &\qquad + \sum_{\ell>k} \left(w_{k\ell}^{\mathrm{TF}} + w_{k\ell}^{\mathrm{TT}} + \min \left\{ (L_{k-1}^\ell - R_{k-l}^\ell)^+ + w_{k\ell}^{\mathrm{FT}}, (L_{k-1}^\ell - R_{k-l}^\ell)^+ + w_{k\ell}^{\mathrm{FF}} \right\} \right) \qquad \text{(Eq-SAT)} \\ &= \sum_{\ell>k} \min \left\{ L_{k-1}^\ell + w_{k\ell}^{\mathrm{FT}}, R_k^\ell + w_{k\ell}^{\mathrm{FF}} \right\} + \mathrm{SAT}_k(\bar{x}) - \mathrm{SAT}_{k-1}(\bar{x}) \end{aligned}$$

The second and third terms of the left-hand side of (Eq-SAT) sum to L_{k-1}^k. Also

$$\mathrm{SAT}_k(\bar{x}) - \mathrm{SAT}_{k-1}(\bar{x}) = L_{k-1}^k + \sum_{\ell>k} \left(w_{k\ell}^{\mathrm{TF}} + w_{k\ell}^{\mathrm{TT}} \right)$$

We can therefore establish (Eq-SAT) by showing that

$$\begin{aligned} &\min \left\{ L_{k-1}^\ell, R_{k-1}^\ell \right\} + \min \left\{ (L_{k-1}^\ell - R_{k-1}^\ell)^+ + w_{k\ell}^{\mathrm{FT}}, (R_{k-1}^\ell - L_{k-1}^\ell)^+ + w_{k\ell}^{\mathrm{FF}} \right\} \\ &\qquad = \min \left\{ L_{k-1}^\ell + w_{k\ell}^{\mathrm{FT}}, R_{k-1}^\ell + w_{k\ell}^{\mathrm{FF}} \right\} \end{aligned}$$

for $\ell > k$. It can be checked that this is an identity. □

6 Problem Formulation: Relaxation

The second phase of problem formulation is to specify when states can be merged, perhaps with an adjustment in transition costs. The goal is to create a relaxed BDD that has limited width but provides a tight bound. In the discussion to follow, we again identify each node u of a BDD with its associated state.

When a layer L_j in the BDD grows too large, we heuristically select a subset M of nodes in the layer to be merged, perhaps by choosing nodes with similar states. The state of the merged nodes is $\oplus(M)$, and the length v of every arc coming into a node $u \in M$ is modified to $\Gamma_M(v, u)$. The process is repeated until $|L_j|$ no longer exceeds the maximum width W.

A relaxed BDD is obtained by using the operators $\oplus$, Γ and a selection heuristic node_select in Algorithm 1, where by convention $L_0 = \varnothing$. The functions $\oplus$, Γ are *valid relaxation operators* if a relaxed BDD results from applying the algorithm with any selection heuristic.

6.1 Formulating the MISP: Relaxation

For the MISP, recall that the state associated with a node is the set of vertices that can still be added to the independent set. States are merged simply by taking their union, so that if $M = \{u_i \mid i \in I\}$, the merged state is $\oplus(M) = \bigcup_{i \in I} u_i$. The transition cost is not changed, so that $\Gamma_M(v, u) = v$ for all v, u. The validity of these relaxation operators is formally proved in Bergman et al. (to appear), but it is clear that taking the union of states does not exclude any paths and therefore results in a relaxed BDD.

Figure 6(b) depicts a relaxed BDD for the MISP on the graph in Figure 1. Nodes u_3 and u_4 are merged to obtain $u' = u_2 \cup u_3 = \{3, 4, 5\}$, which reduces the width of the BDD to 2.

6.2 Formulating the MCP: Relaxation

In the DP model of the MCP, the current state vector s^k contains numbers s^k_ℓ that represent the net benefit of adding vertex ℓ to set T. Recall that we identify each node $u \in L_k$ with the associated state vector s^k. When we merge two nodes u^1 and u^2, we would like the resulting node $u^{\text{new}} = \oplus(\{u, u'\})$ to reflect the values in u and u' as closely as possible, while resulting in a valid relaxation. In particular, path lengths should not decrease. Intuitively, it may seem that $u^{\text{new}}_j = \max\{u^1_j, u^2_j\}$ for each j is a valid relaxation operator, because increasing state values could only increase path lengths. However, this can reduce path lengths as well. It turns out that we can offset any reduction in path lengths by adding the absolute value of the state change to the length of incoming arcs.

We therefore merge the nodes in M as follows. If, for a given ℓ, the states u_ℓ have the same sign for all nodes $u \in M$, we change each u_ℓ to the state with smallest absolute value, and add the absolute value of each change to the length of arcs entering u. When the states u_ℓ differ in sign, we change each u_ℓ to zero and again add the absolute value of the changes to incoming arcs. More precisely, when $M \subset L_k$ we let

$$\oplus(M)_\ell = \left\{ \begin{array}{ll} \min\limits_{u \in M} \{u_\ell\}, & \text{if } u_\ell \geq 0 \text{ for all } u \in M \\ -\min\limits_{u \in M} \{|u_\ell|\}, & \text{if } u_\ell \leq 0 \text{ for all } u \in M \\ 0, & \text{otherwise} \end{array} \right\}, \ \ell = k, \ldots, n$$

$$\Gamma_M(v, u) = v + \sum_{\ell \geq k} \left(|u_\ell| - |\oplus(M)_\ell| \right), \ \text{ all } u \in M \qquad \text{(MCP-relax)}$$

Figure 7 shows the relaxed BDD that results for the example discussed earlier if nodes u_2 and u_3 of the exact BDD are merged.

To show that $\oplus$ and Γ are valid relaxation operators, we rely on the following.

Lemma 4 *Let B be a relaxed BDD generated by Algorithm 1 for an instance $\mathcal{P}$ of the MCP. Suppose we add Δ to one state s^k_ℓ in layer k of B ($\ell \geq k$), and add $|\Delta|$ to the length of each arc entering the node u associated with s^k. If we then recompute layers $k, \ldots, n+1$ of B as in Algorithm 1, the result is a relaxed BDD for $\mathcal{P}$.*

Proof. Let B' the result of recomputing the BDD, and take any $\bar{x} \in \{\text{S}, \text{T}\}^n$. It suffices to show that the path p corresponding to $\bar{x}$ is no shorter in B' than in B. We may suppose p contains u,

because otherwise p has the same length in B and B'. Only arcs of p that leave layers $L_{k-1}, \ldots, L_n$ can have different lengths in B'. The length $v(a)$ of the arc a leaving L_{k-1} becomes $v(a) + |\Delta|$. The states s_ℓ^j along p in B for $j = k, \ldots, n$ become $s_\ell^j + \Delta$ in B', and all other states along p are unchanged. Thus from the formula for transition cost, the length $v(a')$ of the arc a' leaving L_ℓ becomes at least

$$\begin{aligned} v(a') &+ \min\left\{(-(s_\ell^\ell + \Delta))^+, (s_\ell^\ell + \Delta)^+\right\} - \min\left\{(-s_\ell^\ell)^+, (s_\ell^\ell)^+\right\} \\ &\geq v(a') + \min\left\{(-s_\ell^\ell)^+ - \Delta, (s_\ell^\ell)^+ + \Delta\right\} - \min\left\{(-s_\ell^\ell)^+, (s_\ell^\ell)^+\right\} \geq v(a') - |\Delta| \end{aligned}$$

From the same formula, the lengths of arcs leaving L_j for $j > k$ and $j \neq \ell$ cannot decrease. So the length $v(p)$ of p in B becomes at least $v(p) + |\Delta| - |\Delta| = v(p)$ in B'. □

Theorem 5 *Operators $\oplus$ and Γ as defined in (MCP-relax) are valid relaxation operators for the MCP.*

Proof. We can achieve the effect of Algorithm 1 if we begin with the exact BDD, successively alter only one state s_ℓ^k and the associated incoming arc lengths as prescribed by (MCP-relax), and compute the resulting exact BDD after each alteration. We begin with states in L_2 and work down to L_n. In each step of this procedure, we increase or decrease $s_\ell^k = u_\ell$ by $\delta = |u_\ell| - |\oplus(M)_\ell|$ for some $M \subset L_k$, where $\oplus(M)_\ell$ is computed using the states that were in L_k immediately after all the states in L_{k-1} were updated. We also increase the length of arcs into u_ℓ by δ. So we can let $\Delta = \pm\delta$ in Lemma 5 and conclude that each step of the procedure yields a relaxed BDD. □

6.3 Formulating MAX-2SAT: Relaxation

The interpretation of states is very similar for the MCP and MAX-2SAT. We therefore use the same relaxation operators (MCP-relax). The proof of their validity for MAX-2SAT is analogous to the proof of Theorem 7.

Theorem 6 *Operators $\oplus$ and Γ as defined in (MCP-relax) are valid relaxation operators for MAX-2SAT.*

7 Building Restricted BDDs

Restricted BDDs can be constructed in a much simpler way than relaxed BDDs. We need only eliminate nodes from a layer when the layer becomes too large (Bergman et al. submitted). Given a valid DP formulation of a discrete optimization problem, Algorithm 1 constructs a restricted BDD when lines 3–6 are replaced by Algorithm 2. Condition (Res-1) for a restricted BDD is satisfied because the algorithm only deletes solutions, and furthermore, since the algorithm never modifies the states of any nodes that remain, condition (Res-2) must also be satisfied.

We present examples for each of the problems discussed in this paper. For the MISP, Figure 2(c) shows a restricted BDD created by deleting node u_3 in the exact BDD in Figure 6(a). For the MCP, Figure 4(c) shows a restricted BDD created by deleting node u_2 from the exact BDD in Figure 7(a). And finally, for MAX2SAT, Figure 5(c) shows a restricted BDD created by deleting node u_4 in the exact BDD in Figure 8.

Algorithm 2 Restricted BDD Compilation (substitute for lines 3–6 of Algorithm 1)

1: **while** $|L_j| > W$ **do**
2: let $M = \text{state\_select}(L_j)$
3: $L_j \leftarrow L_j \setminus M$

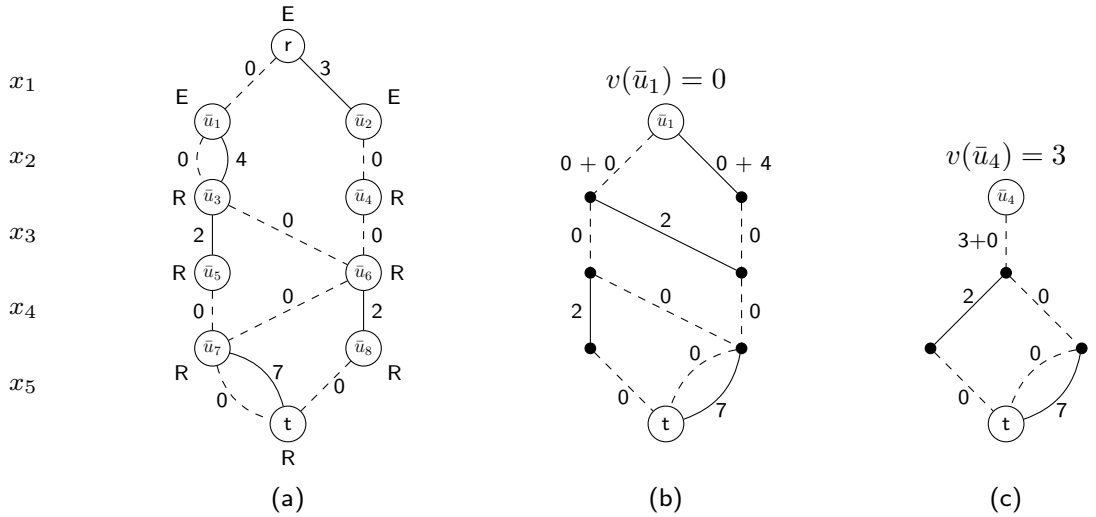


Figure 9: (a) Relaxed BDD for the MISP on the graph in Figure 1 with nodes labeled as exact (E) or relaxed (R); (b) exact BDD for subproblem corresponding to $\bar{u}_1$; (c) exact BDD for subproblem corresponding to $\bar{u}_4$.

8 Branch and Bound

We now present a BDD-based branch-and-bound algorithm. We first define the notion of exact and relaxed nodes and indicate how they can be identified. Then, given a relaxed BDD, we describe a technique that partitions the search space so that relaxed/restricted BDDs can be used to bound the objective function for each subproblem. Finally, we present the branch-and-bound algorithm.

For a given BDD B and nodes $u, u' \in B$ with $\ell(u) < \ell(u')$, we let $B_{uu'}$ be the BDD induced by the nodes that lie on directed paths from u to u' (with the same arc-domains and arc-cost as in B). In particular, $B_{rt} = B$.

8.1 Exact Cutsets

The branch-and-bound algorithm is based on enumerating subproblems defined by nodes in an exact cutset. To develop this idea, let $\bar{B}$ be a relaxed BDD created by Algorithm 1 using a valid DP model of binary optimization problem $\mathcal{P}$. We say that a node $\bar{u}$ in $\bar{B}$ is *exact* if all r–$\bar{u}$ paths in $\bar{B}$ lead to the same state s^j. A *cutset* of $\bar{B}$ is a subset S of nodes of $\bar{B}$ such that any r–t path of $\bar{B}$ contains at least one node in S. We call a cutset *exact* if all nodes in S are exact.

As an illustration, Figure 9(a) duplicates the relaxed BDD $\bar{B}$ from Figure 2 and labels the nodes labeled exact (E) or relaxed (R). Node $\bar{u}_4$ in $\bar{B}$ is an exact node because all incoming paths (there is only one) lead to the same state $\{4, 5\}$. Node $\bar{u}_3$ is relaxed because the two incoming paths represent partial solutions $(x_1, x_2) = (0, 0)$ and $(0, 1)$ that lead to different states, namely $\{3, 4, 5\}$

and $\{5\}$, respectively. Nodes $\bar{u}_1$ an $\bar{u}_4$ form one possible exact cutset of $\bar{B}$.

We now show that an exact cutset provides an exhaustive enumeration of subproblems. If B is an exact BDD for binary optimization problem $\mathcal{P}$, and let $v^*(B_{uu'})$ be the length of a longest u–u' path in $B_{uu'}$. For a node u in B, we define $\mathcal{P}|_u$ to be the restriction of $\mathcal{P}$ whose feasible solutions correspond to r–t paths of B that contain u. Recall that $z^*(\mathcal{P})$ is the optimal value of $\mathcal{P}$.

Lemma 7 *If B is an exact BDD for $\mathcal{P}$, then for any node u in B,*

$$v^*(B_{ru}) + v^*(B_{ut}) = z^*(\mathcal{P}|_u)$$

Proof. $z^*(\mathcal{P}|_u)$ is the length of a longest r–t path of B that contains u, and any such path has length $v^*(B_{ru}) + v^*(B_{ut})$. □

Theorem 8 *Let $\bar{B}$ be a relaxed BDD created by Algorithm 1 using a valid DP model of binary optimization problem $\mathcal{P}$, and let S be an exact cutset of $\bar{B}$. Then*

$$z^*(\mathcal{P}) = \max_{u \in S} \{z^*(\mathcal{P}|_u)\}$$

Proof. Let B be the exact BDD for $\mathcal{P}$ created using the same DP model. Because each node $\bar{u} \in S$ is exact, it has a corresponding node u in B (i.e., a node associated with the same state), and S is a cutset of B. Thus

$$z^*(\mathcal{P}) = \max_{u \in S} \{v^*(B_{ru}) + v^*(B_{ut})\} = \max_{u \in S} \{\mathcal{P}|_u\}$$

where the second equation is due to Lemma 10. □

8.2 Enumeration of Subproblems

We solve a binary optimization problem $\mathcal{P}$ by a branching procedure in which we enumerate a set of subproblems $\mathcal{P}|_u$ each time we branch, where u ranges over the nodes in an exact cutset of the current relaxed BDD. We build a relaxed BDD and a restricted BDD for each subproblem to obtain upper and lower bounds, respectively.

Suppose u is one of the nodes on which we branch. Because u is an exact node, we have already constructed an exact BDD B_{ru} down to u, and we know the length $v^*(u) = v^*(B_{ru})$ of a longest path in B_{ru}. We can obtain an upper bound on $z^*(\mathcal{P}|_u)$ by computing a longest path length $v^*(B_{ut})$ in a relaxed BDD $\bar{B}_{ut}$ with root value $v^*(u)$. To build the relaxation $\bar{B}_{ut}$, we start the execution of Algorithm 1 with $j = \ell(u)$ and root node u, where the root value is $v_r = v^*(u)$. We can obtain a lower bound on $z^*(\mathcal{P}|_u)$ in a similar fashion, except that we use a restricted rather than a relaxed BDD.

The branch-and-bound algorithm is presented in Algorithm 3. We begin with a set $Q = \{r\}$ of open nodes consisting of the initial state r of the DP model. Then, while open nodes remain, we select a node u from Q. We first obtain a lower bound on $z^*(\mathcal{P}|_u)$ by creating a restricted BDD B'_{ut} as described above, and we update the incumbent solution z_{opt}. If B'_{ut} is exact (i.e., $|L_j|$ never exceeds W in Algorithm 2), there is no need for further branching at node u. This is analogous to obtaining an integer solution in traditional branch and bound. Otherwise we obtain an upper bound on $z^*(\mathcal{P}|_u)$ by building a relaxed BDD $\bar{B}_{ut}$ as described above. If we cannot prune the search using this bound, we identify an exact cutset S of $\bar{B}_{ut}$ and add the nodes in S to Q. Because S is

Algorithm 3 Branch-and-Bound Algorithm

initialize $Q = \{r\}$, where r is the initial DP state
let $z_{\text{opt}} = -\infty$, $v^*(r) = 0$
while $Q \neq \varnothing$ **do**
  $u \leftarrow$ select_node(Q), $Q \leftarrow Q \setminus \{u\}$
  create restricted BDD B'_{ut} using Algorithm 1 with root u and $v_r = v^*(u)$
  if $v^*(B_{ut}) > z_{\text{opt}}$ **then**
    $z_{\text{opt}} \leftarrow v^*(B')$
  if B'_{ut} is not exact **then**
    create relaxed BDD $\bar{B}_{ut}$ using Algorithm 1 with root u and $v_r = v^*(u)$
    if $v^*(\bar{B}_{ut}) > z_{\text{opt}}$ **then**
      let S be an exact cutset of $\bar{B}_{ut}$
      for all $u' \in S$ **do**
        let $v^*(u') = v^*(u) + v^*(\bar{B}_{uu'})$, add u' to Q
return z_{opt}

exact, for each $u' \in S$ we know that $v^*(u') = v^*(u) + v^*(\bar{B}_{uu'})$. The search terminates when Q is empty, at which point the incumbent solution is optimal by Theorem 12.

As an example, consider again the relaxed BDD $\bar{B}$ in Figure 9(a). The longest path length in this graph is $v^*(\bar{B}) = 13$, an upper bound on the optimal value. Suppose that we initially branch on the exact cutset $\{\bar{u}_1, \bar{u}_4\}$, for which we have $v(\bar{u}_1) = 0$ and $v(\bar{u}_4) = 3$. We wish to generate restricted and relaxed BDDs of maximum width 2 for the subproblems. Figure 9 (b) shows a restricted BDD $\bar{B}_{\bar{u}_1 t}$ for the subproblem at $\bar{u}_1$, and Figure 9 (c) shows a restricted BDD $\bar{B}_{\bar{u}_4 t}$ for the other subproblem. As it happens, both BDDs are exact, and so no further branching is necessary. The two BDDs yield bounds $v^*(\bar{B}_{\bar{u}_1 t}) = 11$ and $v^*(\bar{B}_{\bar{u}_4 t}) = 10$, respectively, and so the optimal value is 11.

8.3 Selecting an Exact Cutset

Given a relaxed BDD, there are many exact cutsets. Here we present three such cutsets and experimentally evaluate them in Section 10.

- *Traditional branching (TB).* Branching normally occurs by selecting some variable x_j and branching on $x_j = 0/1$. Using the exact cutset $S = L_2$ has the same effect. Traditional branching therefore uses the shallowest possible exact cutset for some variable ordering.

- *Last exact layer (LEL).* For a relaxed BDD $\bar{B}$, define the *last exact layer* of $\bar{B}$ to be the set of nodes $\text{LEL}(\bar{B}) = L_{j'}$, where j' is the maximum value of j for which each node in L_j is exact. In the relaxed BDD $\bar{B}$ of Figure 9(a), $\text{LEL}(\bar{B}) = \{\bar{u}_1, \bar{u}_2\}$.

- *Frontier cutset (FC).* For a relaxed BDD $\bar{B}$, define the *frontier cutset* of B to be the set of nodes

$$\text{FC}(\bar{B}) = \big\{u \text{ in } \bar{B} \mid u \text{ is exact and } b_0(u) \text{ or } b_1(u) \text{ is relaxed}\big\}$$

  In the example of Figure 9(a), $\text{FC}(\bar{B}) = \{\bar{u}_1, \bar{u}_4\}$. A frontier cutset is an exact cutset, due to the following.

Lemma 9 *If $\bar{B}$ is a relaxed BDD that is not exact, then* $\mathrm{FC}(\bar{B})$ *is an exact cutset.*

Proof. By the definition of a frontier cutset, each node in the cutset is exact. We need only show that each solution $x \in \mathrm{Sol}(\bar{B})$ contains some node in $\mathrm{FC}(\bar{B})$. But the path p corresponding to x ends at t, which is relaxed because $\bar{B}$ is not exact. Since the root r is exact, there must be a first relaxed node u in p. The node immediately preceding this node in p is in $\mathrm{FC}(\bar{B})$, as desired. □

9 Modeling Advantages

The recursive formulation of BDD-based models provides a good deal of modeling flexibility, because there is no need for linear, convex, or inequality-based formulations. Any objective function or side constraint that can expressed in terms of the current state and control can be modeled, either in closed form or by subroutine call. While the three problems discussed above already have simple and obvious IP models, they were deliberately chosen for this reason, as it makes comparison with IP more straightforward. In other cases, a recursive formulation can provide a natural model when there is no obvious IP formulation.

Sequencing problems provide good examples. In a simple one-machine scheduling problem, for instance, each job ℓ has processing time time p_ℓ and deadline d_ℓ, where the objective is to minimize total tardiness. At least six IP models have been proposed for the problem, surveyed by Baker and Keller (2010). However, there is a natural recursive model that accommodates a variety of side constraints with no convenient IP formulation. The control variable x_j is the jth job processed (so that there is no need for 0-1 variables). This results in a multivalued decision diagram (MDD) in which the state (S_i, f_i) associated with each node u_i on level j consists of the set S_i of jobs so far processed and the finish time f_j of the last job processed so far. The arcs leaving u_i correspond to the vertices not in S_i. Selecting $x_j = \ell$ effects a transition from (S_i, f_i) to $(S_i \cup \{\ell\}, f_i + p_\ell)$, and the corresponding arc cost is the tardiness $t_\ell = \max\{0, f_i + p_\ell - d_\ell\}$ of job ℓ. Merging nodes with states (S_i, f_i) and (S_k, f_k) in a relaxed MDD yields state $(S_i \cap S_k, \min\{f_i, f_k\})$, where f_i is now interpreted as the earliest finish time. Recursive MDD-based models similar to this have achieved excellent computational results for several sequencing problems (Andersen et al. 2007, Cire and van Hoeve to appear).

The recursive model readily accommodates any side constraint or objective function that can be defined in terms of (S_i, f_i) and x_j. For example, we can shut down the machine for maintenance in the interval $[a, b]$ by transitioning from finish time state f_i to state $f_i + p_\ell + b - a$, rather than to $f_i + p_\ell$, when $a - p_\ell \leq f_i < b$. Release times are also easily enforced. In addition, processing job ℓ may require that certain components already have been fabricated in the processing of previous jobs. We simply exclude the control $x_j = \ell$ when the jobs in S_i do not yield the necessary components. Such side constraints actually make the problem easier by simplifying the MDD. A wide variety of objective functions are also possible. For example, the cost of processing job ℓ may depend on which jobs have already been processed, perhaps again due to common components. We can let the cost associated with control $x_j = \ell$ be any desired function $c_\ell(S_i)$, perhaps evaluated by table lookup. Or cost could be an arbitrary function $c_\ell(t_\ell)$ of tardiness, such as a step function, or a function of both S_i and f_i.

Another modeling illustration is provided by the minimum bandwidth and linear arrangement problems, both of which are defined on an undirected graph $G = (V, E)$ of n vertices. The minimum

bandwidth problem assigns distinct positions $y_j \in \{1,\dots,n\}$ to the vertices j so as to minimize $\max_{(j,k)\in E}\{z_{jk}\}$, where $z_{jk} = |y_j - y_k|$ is the *label* assigned to edge (j,k). The linear arrangement problem differs only in its objective function, which is to minimize $\sum_{(j,k)\in E}\{z_{jk}\}$. It is not obvious how to write IP models for these problems, and Caprara et al. (2011) observe that no useful IP models are known. A recursive model, however, can be formulated as follows. The control variables are the edge labels z_{jk} (plus variable y_1 corresponding to the root node of the MDD). The state at an MDD node on the layer corresponding to z_{jk} is a tuple $(D_1,\dots,D_n)$, where D_j is the current domain of y_j. At the root node, each $D_j = \{1,\dots,n\}$. The arcs leaving a given node in layer z_{jk} of the MDD correspond to all values of $|\ell - \ell'|$ for which $(\ell,\ell') \in D_j \times D_k$. In the linear arrangement problem, the cost on an arc is simply the corresponding value $\bar{z}_{jk}$ assigned to z_{jk}. The arc leads to state $(D'_1,\dots,D'_n)$, where D'_j contains all positions in D_j that have distance $\bar{z}_{jk}$ from some position in D_k, and similarly for D'_k. Also, $D'_i = D_i$ for $i \neq j,k$. The minimum bandwidth problem is only slightly different, as it seeks the r–t path with the smallest maximum arc cost, rather than simply the shortest path. We therefore add a state variable c representing the largest cost so far, so that the arc cost is $\bar{z}_{jk}$ if $\bar{z}_{jk} > c$ and zero otherwise. A variety of side constraints can be enforced, as in other sequencing problems. In the relaxed MDD, states $(D_1,\dots,D_n,c)$ and $(D'_1,\dots,D'_n,c')$ are merged to yield $(D_1 \cup D'_1,\dots,D_n \cup D'_n,\min\{c,c'\})$. A good deal of domain reduction is also possible.

In general, an MDD-based model requires that an overall recursive structure be identified, which is not necessary for an IP model. However, once the recursion is specified, a wide variety of constraints and objectives can be easily expressed in terms of the control and state variables. The availability of state-dependent objective functions is a particularly powerful feature. Furthermore, an MDD model *allows* one to identify and exploit recursive structure, which can be an advantage over an IP model. The recursion also indicates a natural variable ordering for branching, information that an IP model ordinarily does not provide. One must design the node merger mechanism carefully to obtain a good MDD relaxation, but one must likewise formulate an IP model carefully to obtain a good linear relaxation, perhaps by reformulation or addition of redudant constraints and auxiliary variables. In sum, MDD-based optimization provides the modeling flexibility of a dynamic programming formulation without the burden of enumerating the exponential state space that often results from such a formulation.

10 Computational Results

Since we propose BDD-based branch-and-bound as a general discrete optimization method, it is appropriate to measure it against an existing general-purpose method. We compared BDDs with a state-of-the-art IP solver, inasmuch as IP is generally viewed as the most highly-developed general-purpose solution technology for discrete optimization.

Like IP, a BDD-based method requires several implementation decisions, chief among which are the following:

- *Maximum width*: Wider relaxed BDDs provide tighter bounds but require more time to build. For each subproblem in the branch-and-bound procedure, we set the maximum width W equal to the number of variables whose value has not yet been fixed.
- *Node selection for merger*: The selection of the subset M of nodes to merge during the construction of a relaxed BDD (line 4 of Algorithm 1) likewise affects the quality of the bound

(Bergman et al. 2011, 2012, submitted). We use the following heuristic. After constructing each layer L_j of the relaxed BDD, we rank the nodes in L_j according to a rank function rank(u) that is specified in the DP model with the state merging operator $\oplus$. We then let M contain the lowest-ranked $|L_j| - W$ nodes in L_j.

- *Variable ordering*: Much as branching order has a significant impact on IP performance, the variable ordering chosen for the layers of the BDD can affect branching efficiency and the tightness of the BDD relaxation (Bergman et al. 2012). We describe below the variable ordering heuristics we used for the three problem classes.

- *Search node selection*: We must also specify the next node in the set Q of open nodes to be selected during branch and bound (Algorithm 3). We select the node u with the minimum value $v^*(u)$.

The tests were run on an Intel Xeon E5345 with 8GB RAM. The BDD-based algorithm was implemented in C++. The commercial IP solver CPLEX 12.4 was used for comparison. Default settings, including presolve, were used for CPLEX unless otherwise noted. No presolve routines were used for the BDD-based method.

10.1 Results for the MISP

We first specify the key elements of the algorithm that we used for the MISP. Node selection for merger is based on the rank function rank(u) = $v^*(u)$. We used the variable ordering heuristic in Bergman et al. (to appear): after selecting the first $j-1$ variables and forming layer L_j, we choose vertex j as the vertex that belongs to the fewest number of states in L_j. We used FC cutsets for all MISP tests.

For graph $G = (V, E)$, a standard IP model for the MISP is

$$\max \left\{ \sum_{i \in V} x_i \;\middle|\; x_i + x_j \leq 1, \text{ all } (i,j) \in E;\ x_i \in \{0,1\}, \text{ all } i \in V \right\} \tag{1}$$

A tighter linear relaxation can be obtained by pre-computing a clique cover $\mathcal{C}$ of G and using the model

$$\max \left\{ \sum_{i \in S} x_i \;\middle|\; x_i \leq 1, \text{ all } S \in \mathcal{C};\ x_i \in \{0,1\}, \text{ all } i \in V \right\} \tag{2}$$

We refer to this as the *tight* MISP formulation. The clique cover $\mathcal{C}$ is computed using a greedy procedure as follows. Starting with $\mathcal{C} = \varnothing$, let clique S consist of a single vertex v with the highest positive degree in G. Add to S the vertex with highest degree in $G \setminus S$ that is adjacent to all vertices in S, and repeat until no more additions are possible. At this point, add S to $\mathcal{C}$, remove from G all the edges of the clique induced by S, update the vertex degrees, and repeat the overall procedure until G has no more edges.

We begin by reporting results on randomly generated graphs. We generated random graphs with $n \in \{250, 500, \ldots, 1750\}$ and density $p \in \{0.1, 0.2, \ldots, 1\}$ (10 graphs per n, p configuration) according to the Erdös-Rényi model $G(n, p)$ (where each edge appears independently with probability p).

Figure 10 depicts the results. The solid lines represent the average percent gap for the BDD-based technique after 1800 seconds, one line per value of n, and the dashed lines depict the same

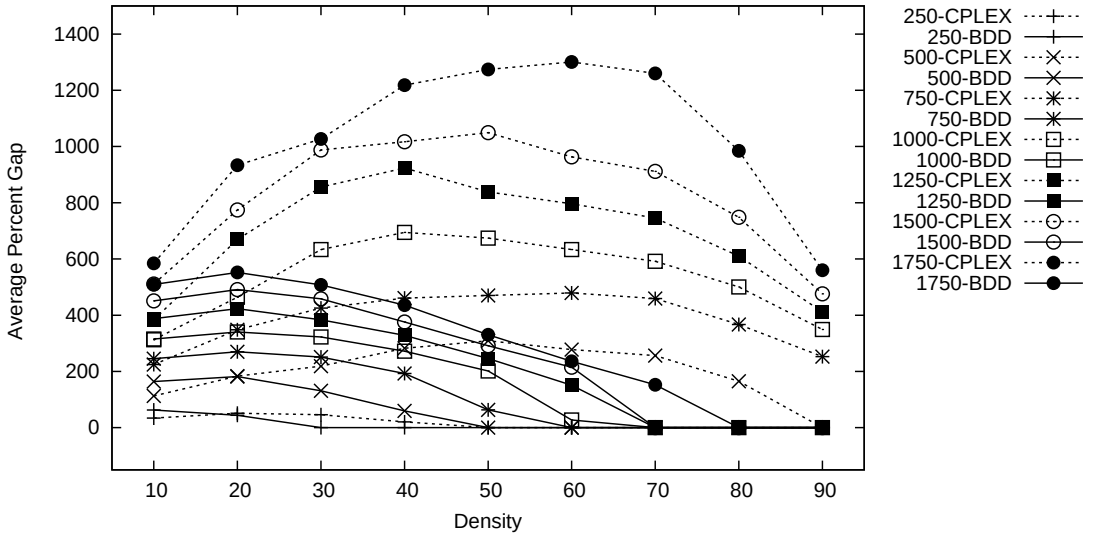


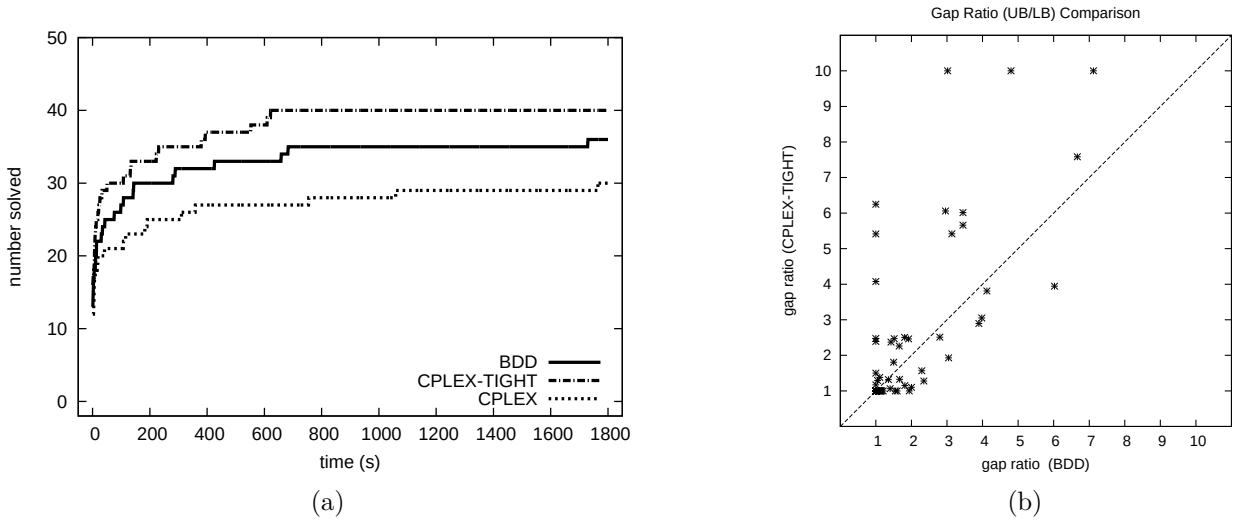
Figure 10: Average percent gap on randomly generated MISP instances.

statistics for the integer programming solver using the tighter, clique model, only. It is clear that the BDD-based algorithm outperforms CPLEX on dense graphs, solving all instances tested with density 80% or higher, and solving almost all instances, except for the largest, with density equal 70%, whereas the integer programming solver could not close any but the smallest instances (with $n = 250$) at these densities.

CPLEX outperformed the BDD technique for the sparsest graphs (with $p = 10$), but only for the small values of n. As n grows, we see that the BDD-based algorithm starts to outperform CPLEX, even on the sparsest graphs, and that the degree to which the ending percent gaps increase as n grows is more substantial for CPLEX than it is for the BDD-based algorithm.

We also tested on the 87 instances of the maximum clique problem in the well-known DIMACS benchmark set (http://cs.hbg.psu.edu/txn131/clique.html). The MISP is equivalent to the maximum clique problem on the complement of the graph.

Figure 11 shows a time profile comparing BDD-based optimization with CPLEX performance for the standard and tight IP formulations. The BDD-based algorithm is superior to the standard IP formulation but solved 4 fewer instances than the tight IP formulation after 30 minutes. However, fewer than half the instances were solved by any method. The relative gap (upper bound divided by lower bound) for the remaining instances therefore becomes an important factor. A comparison of the relative gap for BDDs and the tight IP model appears in Fig. 11(b), where the relative gap for CPLEX is shown as 10 when it found no feasible solution. Points above the diagonal are favorable to BDDs. It is evident that BDDs tend to provide significantly tighter bounds. There are several instances for which the CPLEX relative gap is twice the BDD gap, but no instances for which the reverse is true. In addition, CPLEX was unable to find a lower bound for three of the largest instances, while BDDs provided bounds for all instances.



(a) (b)

Figure 11: Results on 87 MISP instances for BDDs and CPLEX. (a) Number of instances solved versus time for the tight IP model (top line), BDDs (middle), standard IP model (bottom). (b) End gap comparison after 1800 seconds.

10.2 Results for the MCP

We generated random instances of the MCP as follows. For $n \in \{30, 40, 50\}$ and $p \in \{0.1, 0.2, \ldots, 1\}$, we again generated random graphs (10 per n, p configuration). The weights of the edges generated were drawn uniformly from $[-1, 1]$.

We let the rank of a node $u \in L_j$ associated with state s^j be

$$\text{rank}(u) = v^*(u) + \sum_{\ell=j}^{n} \left| s_\ell^j \right|$$

We order the variables x_j according to the sum of the lengths of the edges incident to vertex j. Variables with the largest sum are first in the ordering.

A traditional IP formulation of the MCP introduces a 0–1 variable y_{ij} for each edge $(i, j) \in E$ to indicate whether this edge crosses the cut. The formulation is

$$\min \left\{ \sum_{(i,j)\in E} w_{ij} y_{ij} \;\middle|\; \begin{Bmatrix} y_{ij} + y_{ik} + y_{jk} \le 2 \\ y_{ij} + y_{ik} \ge y_{jk} \end{Bmatrix} \text{ all } i, j, k \in \{1, \ldots, n\};\ y_{ij} \in \{0, 1\}, \text{all } (i, j) \in E \right\}$$

We first consider instances with $n = 30$ vertices, all of which were solved by both BDDs and IP within 30 minutes. Figure 12 shows average solution time for CPLEX and the BDD-based algorithm, using both LEL and FC cutsets for the latter. We tested CPLEX with and without presolve because presolve reduces the model size substantially. We find that BDDs with either type of cutset are substantially faster than CPLEX, even when CPLEX uses presolve. In fact, the LEL solution time for BDDs is scarcely distinguishable from zero in the plot. The advantage of BDDs is particularly great for denser instances.

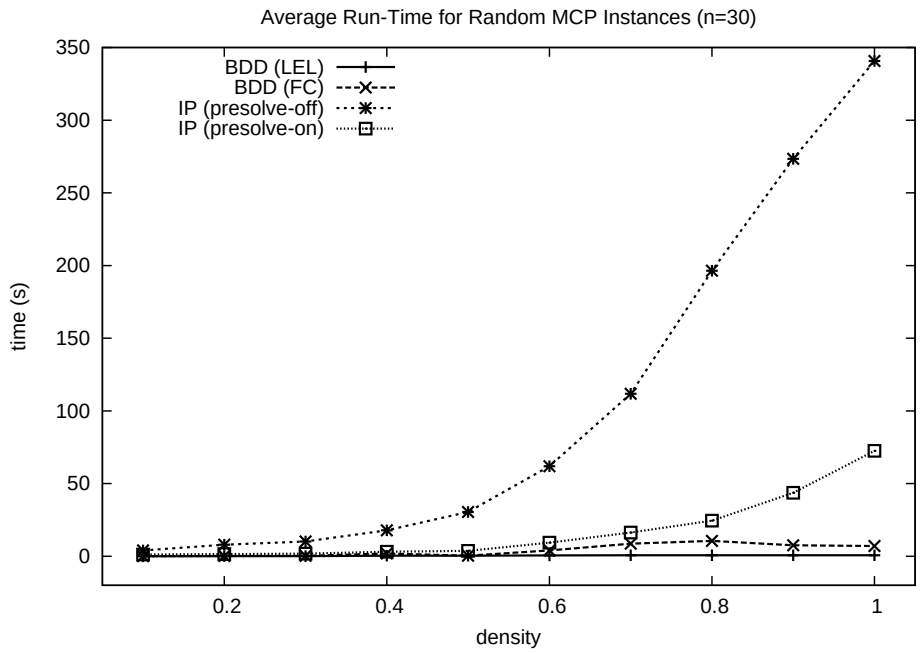


Figure 12: Average solution time for MCP instances ($n = 30$ vertices) using BDDs (with LEL and FC cutsets) and CPLEX (with and without presolve). Each point is the average of 10 random instances.

Results for $n = 40$ vertices appear in Figure 13. BDDs with LEL are consistently superior to CPLEX, solving more instances after 1 minute and after 30 minutes. In fact, BDD solved all but one of the instances within 30 minutes, while CPLEX with presolve left 17 unsolved.

Figure 14(a) shows time profiles for 100 instances with $n = 50$ vertices. The profiles for CPLEX (with presolve) and BDDs (with LEL) are roughly competitive, with CPLEX marginally better for larger time periods. However, none of the methods could solve even a third of the instances, and so the gap for the remaining instances becomes important. Figure 14(b) shows that the average percent gap (i.e., $100(\text{UB} - \text{LB})/\text{LB}$) is much smaller for BDDs on denser instances, and comparable on sparser instances, again suggesting greater robustness for a BDD-based method relative to CPLEX. In view of the fact that CPLEX benefits enormously from presolve, it is conceivable that BDDs could likewise profit from a presolve routine.

We also tested the algorithm on the g-set, a classical benchmark set, created by the authors in Helmberg and Rendl (1997), which has since been used extensively for computational testing on algorithms designed to solve the MCP. The 54 instances in the benchmark set are large, each having at least 800 vertices. The results appear in Table 1 only for those instances for which the BDD-based algorithm was able to improve upon the best known integrality gaps. For the instances with 1% density or more, the integrality gap provided by the BDD-based algorithm is about an order-of-magnitude worse than the best known integrality gaps, but for these instances (which are among the sparsest), we are able to improve on the best known gaps through proving tighter relaxation bounds and identifying better solutions than have ever been found.

The first column provides the name of the instance. The instances are ordered by density, with the sparsest instances reported appearing at the top of the table. We then present the upper bound

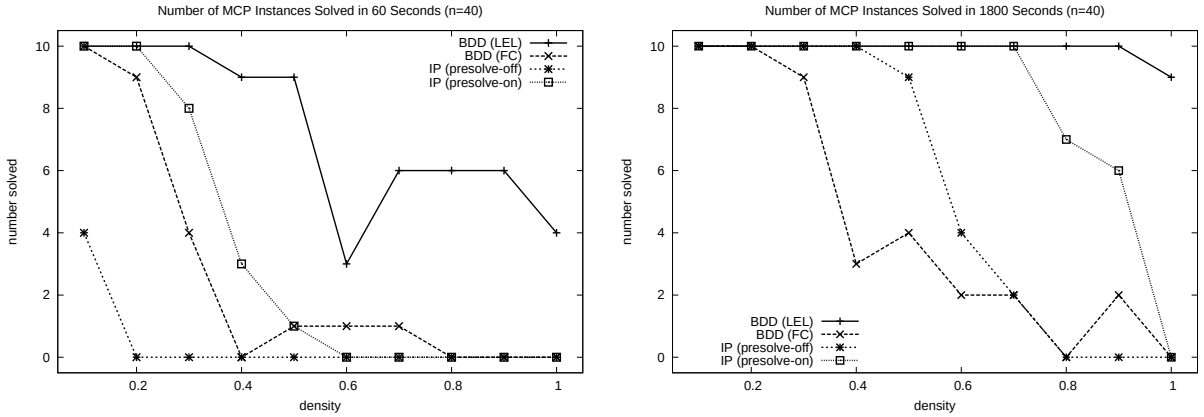
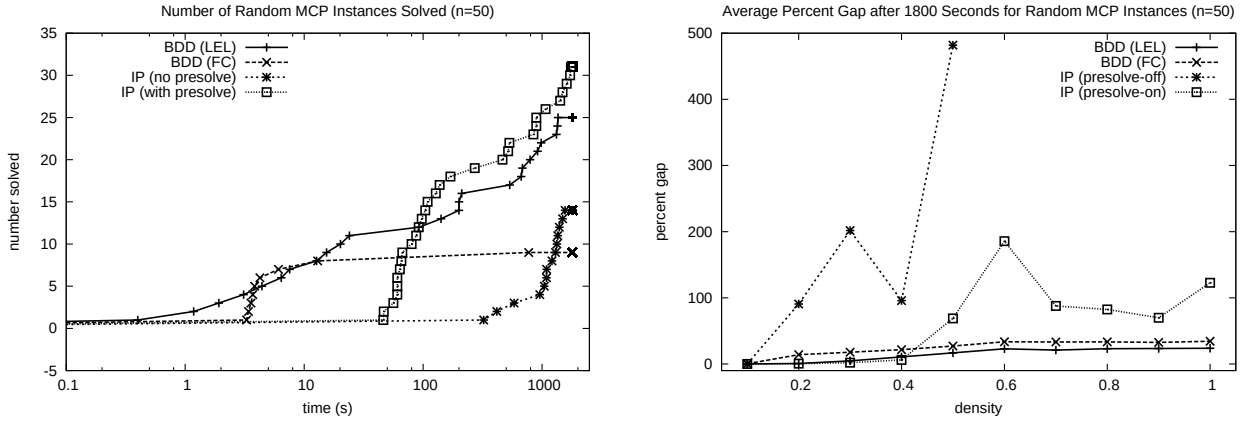


Figure 13: Number of MCP instances with $n = 40$ vertices solved after 60 seconds (left) and 1800 seconds (right), versus graph density, using BDDs (with LEL and FC cutsets) and CPLEX (with and without presolve).



(a) (b)

Figure 14: (a) Time profile for 100 MCP instances with $n = 50$ vertices, comparing BDDs (with LEL and FC cutsets) and CPLEX (with and without presolve). (b) Percent gap versus density after 1800 seconds, where each point is the average over 10 random instances.

(UB) and lower bound (LB), after one hour of computation time, for the BDD-based algorithm, follow by the best known (BK) upper bound and lower bound that we could find in the literature. In the final columns, we record the previously best known percent gap and the new percent gap, where the decrease is due to the improvements from the BDD-based algorithm. Finally, we present the reduction in percent gap obtained.

For three instances (g32, g33, and g34), better solutions were identified by the BDD-based algorithm than have ever been found by any technique, with an improvement in objective function value of 12, 4, and 4, respectively. In addition, for four instances (g50, g33, g11, and g12) better upper bounds were proven than were previously known, reducing the best known upper bound by 89.18, 1, 60, and 5, respectively. For these instances, the reduction in the percent gap is shown in the last column. Most notably, for g50 and g11, the integrality gap was significantly tightened (82.44 and 95.24 percent reduction, respectively). As the density grows, however, the BDD-based algorithm is not able to compete with other state-of-the-art techniques, yielding substantially worse solutions and relaxation bounds than the best known values.

We note here that the BDD-based technique is a general branch-and-bound procedure, whose application to the MCP is only specialized through the DP model that is used to calculate states and determine transition costs. This general technique was able to improve upon best known solutions obtained by heuristics and exact techniques specifically designed to solve the MCP. And so, although the technique is unable to match the best known objective function bounds for all instances, identifying best known solution via this general purpose technique is an indication of the power of the algorithm.

Table 1: G-Set Computational Results

Instance	BDD(UB)	BDD(LB)	BK(UB)	BK(LB)	BK(%gap)	NewBK(%gap)	%ReductionInGap
g50	**5899**	5880	5988.18	5880	1.84	**0.32**	**82.44**
g32	1645	**1410**	1560	1398	11.59	**10.64**	**8.20**
g33	**1536**	**1380**	1537	1376	11.7	**11.30**	**3.39**
g34	1688	**1376**	1541	1372	12.32	**11.99**	**2.65**
g11	**567**	564	627	564	11.17	**0.53**	**95.24**
g12	**616**	556	621	556	11.69	**10.79**	**7.69**

10.3 Results for MAX-2SAT

For the MAX-2SAT problem, we created random instances with $n \in \{30, 40\}$ variables and density $d \in \{0.1, 0.2, \ldots, 1\}$. We generated 10 instances for each pair (n, d), with each of the $4 \cdot \binom{n}{2}$ possible clauses selected with probability d and, if selected, assigned a weight drawn uniformly from $[1, 10]$.

We used the same rank function as for the MCP, and we ordered the variables in ascending order according to the total weight of the clauses in which the variables appear.

We formulated the IP using a standard model. Let clause i contain variables $x_{j(i)}$ and $x_{k(i)}$. Let x_j^i be x_j if x_j is posited in clause i, and $1 - x_j$ if x_j negated. Let δ_i be a 0-1 variable that will be forced to 0 if clause i is unsatisfied. Then if there are m clauses and w_i is the weight of clause i, the IP model is

$$\max \left\{ \sum_{i=1}^{m} w_i \delta_i \;\middle|\; x^i_{j(i)} + x^i_{k(i)} + (1 - \delta_i) \geq 1, \text{all } i;\ x_j, \delta_i \in \{0, 1\},\ \text{all } i, j \right\}$$

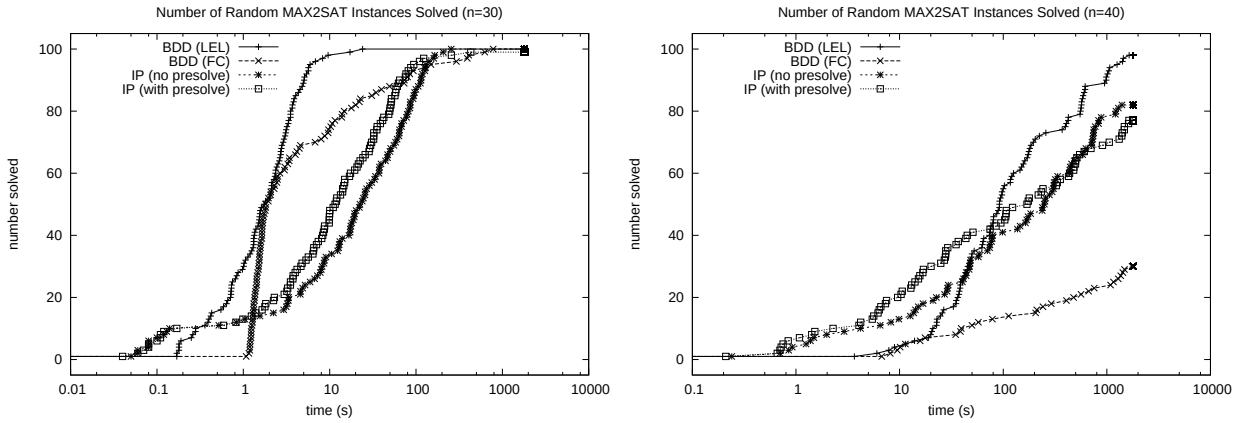


Figure 15: Time profile for 100 MAX-2SAT instances with $n = 30$ variables (left) and $n = 40$ variables (right), comparing BDDs (with LEL and FC cutsets) and CPLEX (with and without presolve).

Figures 15 shows the time profiles for the two size classes. BDDs with LEL are clearly superior to CPLEX for $n = 30$. When $n = 40$, BDDs prevail over CPLEX as the available solving time grows. In fact, BDDs solve all but 2 of the instances within 30 minutes, while CPLEX leaves 17 unsolved using no presolve, and 22 unsolved using presolve.

11 Conclusion

We presented a new branch-and-bound optimization method in which decision diagrams provide discrete relaxations, a primal heuristic, and a framework for branching. The method uses a dynamic programming formulation of the problem and therefore has no need of an inequality model, even though it does not solve the problem with dynamic programming. The branching scheme enumerates pools of partial solution rather than individual assignments of values to variables.

A general optimization method cannot be fully evaluated, nor its potential fully realized, without a significant period of experimentation and development. Integer programming, for example, evolved over decades, during which it improved by several orders of magnitude. Nonetheless, our initial experience suggests that a prototype BDD-based solver is competitive with, or superior to, the latest integer programming technology on three problem classes that are readily formulated for an integer solver.

Unlike integer programming, the BDD-based method described here has no obvious extension to continuous variables, and it requires that problems be modeled as a dynamic programming recursion. On the other hand, it has no need of integer variables or linear constraints, and the problem formulation may allow one to take advantage of structure that cannot be exploited in other modeling regimes. It may provide an alternative approach to problems with huge inequality models, including models that are too large to load into a linear programming solver. BDDs can also work alongside integer programming by supplying additional upper and lower bounds and even a framework for branching.

Several research questions remain. One is whether the advantage of BDDs is greater on problems that are difficult to model for linear integer programming, such as minimum bandwidth problems or

quadratic assignment problems. A second is whether modeling flexibility can be usefully extended by nonserial dynamic programming formulations, as suggested by Hooker (2013). Finally, one might ask whether BDD methods can be profitably adapted to stochastic problems, perhaps along the lines of Sanner and McAllester (2005), or even to continuous optimization.

References

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

Andersen, H. R., T. Hadžić, J. N. Hooker, P. Tiedemann. 2007. A constraint store based on multivalued decision diagrams. C. Bessière, ed., *Principles and Practice of Constraint Programming (CP 2007)*, *Lecture Notes in Computer Science*, vol. 4741. Springer, 118–132.

Baker, K. R., B. Keller. 2010. Solving the single-machione sequencing problem using integer programming. *Computers and Industrial Engineering* **59** 730–735.

Balasundaram, B., S. Butenko, I. V. Hicks. 2011. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research* **59** 133–142.

Baldacci, R., A. Mingozzi, R. Roberti. 2012. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing* **24** 356–371.

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

Behle, M., F. Eisenbrand. 2007. 0/1 vertex and facet enumeration with BDDs. *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 158–165.

Bergman, D., A. A. Cire, W.-J. van Hoeve, J. N. Hooker. 2012. Variable ordering for the application of BDDs to the maximum independent set problem. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2012)*, *LNCS*, vol. 7298. Springer, 34–49.

Bergman, D., A. A. Cire, W.-J. van Hoeve, J. N. Hooker. to appear. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing* .

Bergman, D., A. A. Cire, W.-J. van Hoeve, T. Yunes. submitted. BDD-based heuristics for binary optimization .

Bergman, D., W.-J. van Hoeve, J. N. Hooker. 2011. Manipulating MDD relaxations for combinatorial optimization. *Proceedings of CPAIOR*, *LNCS*, vol. 6697. 20–35.

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

Caprara, A., A. N. Letchford, J-J. Salazar-González. 2011. Decorous lower bounds for minimum linear arrangement. *INFORMS Journal on Computing* **23** 26–40.

Christofides, N., A. Mingozzi, P. Toth. 1981. State-space relaxation procedures for the computation of bounds to routing problems. *Networks* **11** 145–164.

Cire, A. A., W.-J. van Hoeve. to appear. Multivalued decision diagrams for sequencing problems. *Operations Research* .

Eblen, J. D., C. A. Phillips, G. L. Rogers, M. A. Langston. 2011. The maximum clique enumeration problem: Algorithms, applications and implementations. *Proceedings of the 7th international conference on Bioinformatics research and applications*. ISBRA'11, Springer-Verlag, Berlin, Heidelberg, 306–319.

Edachery, J., A. Sen, F. J. Brandenburg. 1999. Graph clustering using distance-k cliques. *Proceedings of Graph Drawing*, *LNCS*, vol. 1731. Springer-Verlag, 98–106.

Festa, P., P. M. Pardalos, M. G. C. Resende, C. C. Ribeiro. 2002. Randomized heuristics for the max-cut problem. *Optimization Methods and Software* **7** 1033–1058.

Hadžić, T., J. N. Hooker. 2006. Postoptimality analysis for integer programming using binary decision diagrams. Tech. rep., Carnegie Mellon University.

Hadžić, T., J. N. Hooker. 2007. Cost-bounded binary decision diagrams for 0-1 programming. E. Loute, L. Wolsey, eds., *CPAIOR 2007 Proceedings*, *Lecture Notes in Computer Science*, vol. 4510. Springer, 84–98.

Hadžić, T., J. N. Hooker, B. O'Sullivan, P. Tiedemann. 2008a. Approximate compilation of constraints into multivalued decision diagrams. P. J. Stuckey, ed., *Principles and Practice of Constraint Programming (CP 2008)*, *Lecture Notes in Computer Science*, vol. 5202. Springer, 448–462.

Hadžić, T., J. N. Hooker, P. Tiedemann. 2008b. Propagating separable equalities in an MDD store. L. Perron, M. A. Trick, eds., *CPAIOR 2008 Proceedings*, *Lecture Notes in Computer Science*, vol. 5015. Springer, 318–322.

Hager, W. W., Y. Krylyuk. 1999. Graph partitioning and continuous quadratic programming. *SIAM Journal on Discrete Mathematics* **12** 500–523.

Helmberg, C., F. Rendl. 1997. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization* **10** 673–696.

Hoda, S., W.-J. van Hoeve, J. N. Hooker. 2010. A systematic approach to MDD-based constraint programming. *Proceedings of the 16th International Conference on Principles and Practices of Constraint Programming (CP 2010)*, *Lecture Notes in Computer Science*, vol. 6308. Springer, 266–280.

Hooker, J. N. 2006. Discrete global optimization with binary decision diagrams. *GICOLAG 2006*. Vienna, Austria.

Hooker, J. N. 2013. Decision diagrams and dynamic programming. *CPAIOR 2013 Proceedings*. 94–110.

Hu, A. J. 1995. Techniques for efficient formal verification using binary decision diagrams. Thesis CS-TR-95-1561, Stanford University, Department of Computer Science.

Kam, T., T. Villa, R. K. Brayton, A. L. Sangiovanni-Vincentelli. 1998. Multi-valued decision diagrams: Theory and applications. *International Journal on Multiple-Valued Logic* **4** 9–62.

Kell, B., W.-J. van Hoeve. 2013. An MDD approach to multidimensional bin packing. *Proceedings of CPAIOR*, *Lecture Notes in Computer Science*, vol. 7874. Springer, 128–143.

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

Loekito, E., J. Bailey, J. Pei. 2010. A binary decision diagram based approach for mining frequent subsequences. *Knowl. Inf. Syst* **24** 235–268.

Minato, S. 1993. Zero-suppressed BDDs for set manipulation in combinatorial problems. *30th Conference on Design Automation*. IEEE, 272–277.

Mingozzi, A. 2002. State space relaxation and search strategies in dynamic programming. *Proceedings of Abstraction, Reformulation, and Approximation*, *Lecture Notes in Computer Science*, vol. 2371. Springer, 51–51.

Righini, G., M. Salani. 2008. New dynamic programming algorithms for the resource constrained shortest path problem. *Networks* **51** 155–170.

Sanner, S., D. McAllester. 2005. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*. 1384–1390.

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