Incorporating Bounds from Decision Diagrams into Integer Programming

Christian Tjandraatmadja · Willem-Jan van Hoeve

Abstract Decision diagrams have been successfully used to help solve several classes of discrete optimization problems. We explore an approach to incorporate them into integer programming solvers, motivated by the wide adoption of integer programming technology in practice. The main challenge is to map generic integer programming models to a recursive structure that is suitable for decision diagram compilation. We propose a framework that opportunistically constructs decision diagrams for suitable substructures, if present. In particular, we explore the use of a prevalent substructure in integer programming solvers known as the conflict graph, which we show to be amenable to decision diagrams. We use Lagrangian relaxation and constraint propagation to consider constraints that are not represented directly by the substructure. We use the decision diagrams to generate dual and primal bounds to improve the pruning process of the branch-and-bound tree of the solver. Computational results on the independent set problem with side constraints indicate that our approach can provide substantial speedups when conflict graphs are present.

Keywords Integer Programming · Decision Diagrams

Mathematics Subject Classification (2010) 90C10 · 90C35

1 Introduction

Decision diagrams were originally introduced to compactly represent Boolean functions, and have been widely applied to verification and configuration prob-

Christian Tjandraatmadja
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA
E-mail: ctjandra@alumni.cmu.edu
Currently at Google

Willem-Jan van Hoeve
Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA
E-mail: vanhoeve@andrew.cmu.edu

lem [30, 4, 21, 38]. More recently, decision diagrams have been applied to model and solve combinatorial optimization problems, in particular via a stand-alone solver in which relaxed and restricted decision diagrams provide dual and primal bounds as well as a branch-and-bound search strategy [20, 17, 16].

A strength of decision diagrams lies in representing recursive structure embedded in certain discrete optimization problems. Typically, this structure is explicitly modeled by a user through a dynamic programming formulation [15], which is often not readily available when facing a new problem. Instead, it is common for discrete optimization problems to be modeled via integer programming (IP) formulations, due to the effectiveness of mixed-integer programming (MIP) solvers and the capability of IP models to express a variety of problems.

In this paper, we propose a framework to improve the solution process of MIP solvers through the use of relaxed decision diagrams: decision diagrams that represent relaxations of a problem. To achieve this goal, we study two main research questions. The first question is how to construct effective relaxed decision diagrams from generic IP formulations. Provided with a method to do so, the second question is how to use them to aid the MIP solver.

One of the main challenges in constructing relaxed decision diagrams from IP models is that the linear formulation may not give access to the original problem structure. For example, successful applications of decision diagrams in the context of scheduling and routing [22, 28, 33] require a constraint-based representation that is not easy to recognize when presented as a linearized MIP model. It is important to recognize that the effectiveness of decision diagrams is tightly connected to the structure modeled by the dynamic programming (DP) formulations used to construct them. Therefore, the complex structure of real-world problems may present an obstacle for approaches based on decision diagrams. A solution to this challenge is to break these problems apart into structures that are more tractable from the perspective of decision diagrams. For instance, several problems contain set packing constraints – constraints of the form $Ax \leq 1$, where A is a binary matrix and x is a binary vector of variables – which, when isolated, tend to be receptive to approaches based on decision diagrams [17].

This motivates us to develop methods for the case when a known structure exploitable by decision diagrams is partially present in a problem. We propose a framework that builds decision diagrams for classes of constraints present in the problem – which can be viewed as relaxations – and incorporates the remaining constraints via two approaches, Lagrangian relaxation and constraint propagation. This framework is aimed towards generating dual bounds for the problem, which we later use to aid the MIP solver.

While this framework permits any choice of substructure, in this paper we investigate the use of the conflict graph for binary problems [7, 1]. The conflict graph is a common component in modern MIP solvers and represents the pairs of binary variables that cannot take a certain pair of values. It can be viewed as a relaxation of the problem and thus it fits our framework. Moreover, as we later show in this paper, the feasible set of a conflict graph admits a good DP formulation. A benefit of using the conflict graph is that the method requires no additional input from the user other than the IP model itself. Nevertheless, a user could provide a DP formulation of a different substructure of a problem as well.

Although focusing on a specific substructure limits the range of applications, we do not aim to design a method to improve the solution of any arbitrary IP model. Instead, our approach is opportunistic: we only attempt to aid the solution process for a model when there are reasons to believe that decision diagrams can help – for example, when a conflict graph is present and captures a substantial part of the problem. Additional substructures may be incorporated in future research, extending the applicability of this framework.

After constructing the relaxed decision diagrams, we consider the question of how to leverage them to reduce solving times in MIP solvers. Decision diagrams have been used early on to solve integer programs via an independent branch-and-bound mechanism [29] and later to generate cutting planes within MIP solvers [8, 36], including in stochastic [31] and nonlinear [23] settings. In this work, we generate dual bounds from these decision diagrams throughout the branch-and-bound tree in order to identify additional pruning opportunities. In other words, this approach is oriented towards eliminating subproblems that only contain suboptimal solutions. We computationally test this technique on two classes of instances: one in which the entire problem can be expressed as conflict constraints (the independent set problem), and one in which the conflict graph only partially captures it (the independent set problem augmented with knapsack constraints).

The dual bounds we generate in our framework come from solving the following relaxation:

$$\max_{x}\{c^\top x : \hat{A}x \leq \hat{b},\ x \in \operatorname{conv}(S)\}, \tag{1}$$

where $c^\top x$ models the objective function (assuming maximization), S is a relaxation of the feasible set represented by a relaxed decision diagram, and $\hat{A}x \leq \hat{b}$ are constraints that cannot be efficiently represented by the decision diagram in practice.

Along with dual bounds, we also generate primal feasible solutions. Better primal feasible solutions not only improve the pruning process but are also informative for the user if the solving process is terminated before reaching optimality. Conversely, we show how to use primal bounds generated from the MIP solver to speed up the construction of relaxed decision diagrams.

We begin by defining decision diagrams in Section 2 and providing an overall view of the framework in Section 3. Sections 4 and 5 detail two important aspects of the framework: constructing decision diagrams for conflict graphs and handling constraints that are not considered in the decision diagrams. Section 6 discusses techniques related to primal bounds. Finally, Section 7 presents computational results and Section 8 concludes this paper.

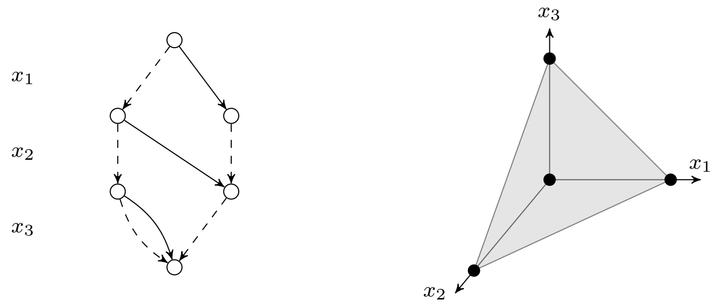


Fig. 1: On the left, a decision diagram that represents the set $\{x \in \{0,1\}^3 : x_1 + x_2 + x_3 \leq 1\}$, pictured on the right. A dashed line indicates an arc with value zero, while a full line indicates an arc with value one. Each of the four paths from the root to the terminal nodes corresponds to a point in the set.

2 Decision Diagrams

In the context of optimization, we can view a *decision diagram* (DD) as a graph that represents the feasible set of a discrete optimization problem. In general, they can represent discrete sets of points or Boolean functions. More formally, a decision diagram is a directed acyclic multigraph in which the nodes and arcs form layers, as illustrated in Figure 1. The arcs of each layer k correspond to assigning some value v to a variable x_k. Parallel arcs are allowed. The first layer has a single root node s and the last layer, a single terminal node t. We assume that all nodes except the root and terminal have at least one incoming arc and one outgoing arc. We denote by *width* the size of the largest layer of the decision diagram.

The feasible set S of a problem is represented through a one-to-one correspondence between each $x \in S$ and each directed path from s to t in the decision diagram. The solution x is represented by the assignments that correspond to the arcs in the path, noting that all variables are represented in the layers. We remark that given an ordering of variables, there exists a unique smallest decision diagram for S, called a *reduced decision diagram* [21].

Given a constructed decision diagram for S, we can efficiently optimize a linear function over S. Due to the correspondence between paths and solutions in S, maximizing a linear function $c^\top x$ entails finding a path on the decision diagram of maximum weight, given weights $c_k v_k$ on each layer-k arc with value v_k. Since a decision diagram is a directed acyclic graph, this can be done in time linear in the number of arcs in the decision diagram.

Representing the feasible set of hard discrete optimization problems will often result in impractically large decision diagrams. To tackle this issue, we consider *relaxed decision diagrams* [5, 20, 16], which are relaxations in the form of decision diagrams – that is, they contain the feasible set but are not required to represent it exactly. Relaxed decision diagrams are usually made smaller than exact ones by merging nodes in a way that avoids removing feasible solutions, but may add infeasible ones. Typically, throughout the construction, the nodes of a layer are merged until a given width limit is satisfied. See e.g. [16] for more details on a construction method for relaxed decision diagrams. A relaxed decision diagram can yield a dual bound (i.e., a lower bound if we are minimizing or an upper bound if we are maximizing) for the problem by optimizing the linear objective function over it, since it is a relaxation [20, 17]. We refer to [15] for an overview of the use of decision diagrams in the context of combinatorial optimization.

3 Framework

A central challenge in designing approaches based on relaxed decision diagrams is to keep them small while still obtaining a good approximation of the problem. We next discuss two main factors that influence the strength of this approximation: the ability to identify equivalent nodes and the form of relaxation.

The power of decision diagrams comes from merging equivalent nodes. Equivalent nodes are those that have the same *completion set*, defined as the set of possible assignments leading to feasible solutions given the assignments made from the root to that node. The problem of *complete equivalence* is to decide whether two nodes are equivalent or not. Unfortunately, deciding whether two nodes are equivalent or not in the context of linear constraints is NP-complete. This is because a special case of node equivalence is deciding whether a node is equivalent to an infeasible node, and deciding integer feasibility for two or more linear constraints is NP-complete. While in practice it is not vital that we merge every pair of equivalent nodes possible, merging as many as possible allows us to focus on other important factors that affect the size of decision diagrams.

Nevertheless, even if we can efficiently identify equivalent nodes, a decision diagram can still grow exponentially large. To manage its size, we must approximate the problem with a tractable relaxation. In this framework, we consider two forms of relaxation: one at the level of decision diagram construction and another at the level of problem constraints.

At the decision diagram level, we construct relaxed decision diagrams using a top-down construction, as done in recent literature [16]. We set a maximum width parameter and whenever a layer has higher width than this parameter, we merge (non-equivalent) nodes until the width is within the maximum. Ensuring a relaxation in this merging process is problem-dependent: a node typically is associated to a state which implicitly encodes its completion set, and merging non-equivalent nodes requires finding a state that encodes a completion set containing the union of the completion sets of the two original nodes. For example, in the independent set problem, a state would be the vertices that can still be selected and the merged state would be the union of the two sets of available vertices from the original states. The criteria for choosing which nodes to merge may depend on the application, but there exist generic rules such as merging nodes with poor objective values. We refer to Chapter 4 of [15] for more details on node merging for relaxed decision diagrams.

At the constraint level, the framework considers a substructure of the problem, such as a subset of constraints of a specific type or, in the case of this work, conflict graphs. Not only may substructures have a more tractable size than the overall problem, but more importantly information about problem structure can significantly benefit the construction of decision diagrams. However, this relaxation can be very weak if it ignores constraints not captured by the substructure. We call such constraints *generic*. This is compensated through the use of Lagrangian relaxation, which can be used with decision diagrams [14]. Moreover, we can partially incorporate them into a decision diagram through the use of constraint propagation. More details are presented in Section 5.

Given that we use a substructure as the basis for our decision diagram, we must choose a structure with good qualities. We balance the following criteria in the choice of structure:

– **Identifiability:** We should be able to efficiently identify and extract the substructure from the problem. While this is trivial if we choose an explicit subset of constraints, we may also consider relaxations that are not explicitly given in the problem.
– **Generality:** The structure should be as generic as possible in order to capture structure within as many applications as possible. In particular, this structure must play a fundamental role in defining the problems we aim to improve upon, as otherwise the bounds generated would be weak.
– **Compactness:** In order to keep the size of the decision diagram compact, the formulation must ideally support efficient equivalence tests that are complete or close to being complete. Moreover, structures with good variable ordering and merging (relaxation) heuristics are desirable. It is well known that variable ordering can have a considerable effect on the size of the decision diagram [21] and likewise the quality of the bound from relaxed decision diagrams [18].

For binary problems, conflict graphs satisfy these three criteria well. First, the task of identifying the conflict graph structure (when present) is already performed by modern MIP solvers, and thus we do not need to be concerned with extracting them. Second, the conflict graph encompasses common constraints such as set packing constraints and simple implications of the form $x_i = v_i \implies x_j = v_j$. These are equivalent to 2-SAT constraints (see, e.g., [6]) as well as implication graphs, as we will discuss in the next section. Observe that if we were to extend the representation to 3-SAT, complete equivalence becomes NP-complete, since the special case of deciding feasibility is also NP-complete for 3-SAT. Third, we will present in Section 4 a DP formulation for the conflict graph that has an efficient complete equivalence test in top-down construction. In addition, we generalize a variable ordering heuristic for the independent set problem, previously shown to perform well in practice [17].

By choosing the conflict graph as a substructure, we limit this work to binary problems. Nevertheless, the framework itself supports any type of structure, as long as we can efficiently build good decision diagrams from them. In this context, structure means any class of constraints that forms a relaxation of the problem. Examples of structures that may work well with decision diagrams, and are defined on the original problem variables, include set partitioning and set packing (as special cases of the conflict graph), set covering [20, 19], and maximum cut problems [16]. When the model representation is extended beyond traditional MIP formulations, one could incorporate structures for single-machine scheduling and routing [22, 28, 33]. More generally, a user may provide a DP formulation of a substructure of a specific problem, or combine multiple classes of constraints. Decomposing a problem with multiple decision diagrams may also be implemented in this framework [11]. These extensions are however beyond the scope of this work.

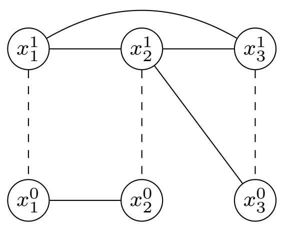
We apply our approach to generate dual bounds from relaxed decision diagrams at certain nodes of the branch-and-bound tree to improve the pruning process. Pruning is performed as usual: if the current primal bound is at least as good as the dual bound generated at a subproblem node, then the subproblem contains no improving solution and does not need to be explored. In this paper, we focus on generating them in small subproblems of the tree.

A summary of the framework is as follows.

1. We select a substructure of the problem from which to construct a relaxed decision diagram – in this paper, we use a conflict graph.
2. We construct a decision diagram, possibly relaxed, using a DP formulation specific to the substructure (Section 4). During construction, we may propagate information from generic constraints (constraints not implied by the substructure) into the decision diagram (Section 5.2).
3. Once the decision diagram is constructed, we apply Lagrangian relaxation in order to further incorporate generic constraints into the bound (Section 5.1).
4. This process yields dual bounds. Primal bounds may also be obtained (Section 6). These bounds are added to certain nodes of the branch-and-bound tree of the MIP solver to help pruning. Nodes with smaller subproblems are prioritized.

4 Decision Diagrams for Conflict Graphs

The conflict graph captures constraints that forbid certain pairs of binary variables from taking specific values. More formally, a conflict graph $G = (V, E)$ is a graph with two vertices per binary variable of the problem. Each vertex corresponds to an assignment of 0 or 1 to the corresponding variable. Denote by x_j^v the node of the conflict graph corresponding to the assignment of v to the variable x_j. We use x_j^{1-v} to denote the node corresponding to the negation of x_j^v. An edge exists between x_i^u and x_j^v if the assignments $x_i = u$



$$x_1 + x_2 + x_3 \leq 1$$

$$x_2 + (1 - x_3) \leq 1$$

$$(1 - x_1) + (1 - x_2) \leq 1$$

$$x_1, x_2, x_3 \in \{0, 1\}$$

Fig. 2: Example of a conflict graph for three binary variables, where x_i^0 and x_i^1 indicate setting x_i to 0 and 1 respectively. On the right, a linear representation of the constraints from the conflict graph.

and $x_j = v$ cannot simultaneously occur in a feasible solution of the problem. Figure 2 illustrates an example of a conflict graph.

Conflict constraints can be inferred in MIP solvers when applying, for instance, bound strengthening or probing during a presolve step. A common use of a conflict graph is to generate cuts [7, 1].

Note that each conflict constraint on x_i^u and x_j^v is equivalent to the constraint $x_i = u \implies x_j = 1 - v$, which is itself equivalent to $x_j = v \implies x_i = 1 - u$. Therefore, we can express conflict constraints as implication constraints by replacing each edge $\{x_i^u, x_j^v\}$ with a pair of directed arcs (x_i^u, x_j^{1-v}) and (x_j^v, x_i^{1-u}). The resulting graph is called an implication graph. Since this conversion can occur in both directions, conflict graphs are equivalent to implication graphs.

Throughout this section, it is more convenient to describe a formulation for the implication graph instead of the conflict graph. Concepts from this formulation can be directly translated to the context of the conflict graph through the above equivalence.

We remark that modern MIP solvers may construct implication graphs for general integer variables instead of binary [1]. However, in this work, we focus on the binary setting.

4.1 Dynamic programming formulation

As described in [15], decision diagrams can be constructed from a dynamic programming model, which provides the state definitions for its nodes, as well as the transition function between nodes. We therefore first provide a dynamic programming formulation for the feasible set of the implication graph – that is, the set of all solutions that satisfy the implication constraints encoded in the graph.

For notational convenience, we assume variables are ordered as $x_1, \ldots, x_n$. The layer j (or stage j in DP terms) contains the states in which we have defined assignments for variables $x_1, \ldots, x_{j-1}$ and seek to assign values to

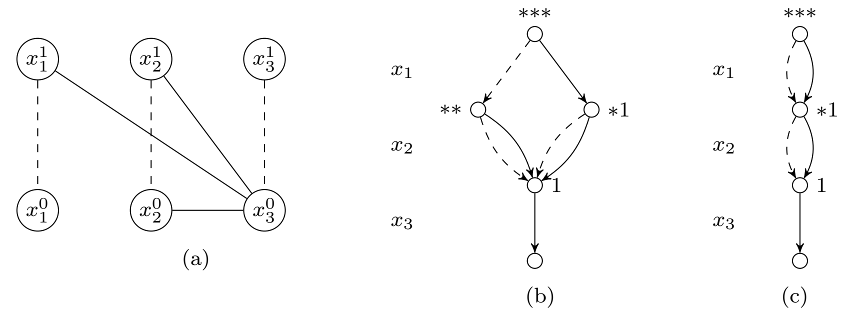


Fig. 3: (a) A conflict graph. (b) The decision diagram that would be obtained by using the DP formulation (IG) as is. The states are depicted as a sequence of symbols representing the domain of each completion variable following the order x_1, x_2, x_3. The symbol is $*$ if the domain is $\{0, 1\}$, 1 if it is $\{1\}$, and 0 if it is $\{0\}$. (c) The decision diagram obtained if we establish domain consistency at the root state, which is always reduced as proved in Theorem 1.

$x_j, \dots, x_n$. The values to be assigned are restricted to the variable's domain, denoted by $\mathcal{D}(x_j)$ for each variable x_j, which for conflict constraints we assume to be $\{0, 1\}$. We denote the infeasible state by $\hat{0}$.

The DP formulation for the implication graph works as follows. Each state at layer j corresponds to domains of the variables $x_j, \dots, x_n$. When we transition by setting the variable x_j to v_j, we remove from the domains all assignments $x_k = v_k$ such that $x_k^{1-v_k}$ is reachable from $x_j^{v_j}$. Here, we say that u is reachable from v if there exists a directed path in the implication graph from u to v. In other words, we take the implied assignments and remove their complements from the domains.

More precisely, the DP formulation of the implication graph, which we denote by (IG) for the remainder of the paper, is defined as follows:

- **State space:** A state s in the state space $\mathcal{S}^j$ of stage j represents a list $(\mathcal{D}(x_j), \mathcal{D}(x_{j+1}), \dots, \mathcal{D}(x_n))$ such that each $\mathcal{D}(x_j)$ represents the domain of the variable x_j, and thus may take the values $\varnothing$, $\{0\}$, $\{1\}$, or $\{0, 1\}$ in this binary setting.
- **Transition function:** Let $\mathcal{D}_s$ be the domain associated to state s. Given an assignment v_j to a variable x_j, denote by $\mathcal{D}'_s(x_k) = \mathcal{D}_s(x_k) \setminus \bar{R}_{k,j,v_j}$, where $\bar{R}_{k,j,v_j}$ is the set of values v_k such that the node $x_k^{1-v_k}$ is reachable from $x_j^{v_j}$ in the implication graph G. The transition function t_j at layer j is defined as:

$$t_j(s, v_j) = \begin{cases} (\mathcal{D}'_s(x_{j+1}), \mathcal{D}'_s(x_{j+2}), \dots, \mathcal{D}'_s(x_n)) \\ \quad \text{if } v_j \in \mathcal{D}_s(x_j) \text{ and } \mathcal{D}'_s(x_k) \neq \varnothing \text{ for all } k = j+1, \dots, n, \\ \hat{0} \quad \text{otherwise.} \end{cases}$$

Figure 3 illustrates a decision diagram constructed from this DP formulation. We next show that this formulation is correct and provides a sufficient condition for complete equivalence that can be efficiently guaranteed.

4.1.1 Correctness

The proposition below shows that (IG) models the implication graph.

Proposition 1 *The DP formulation (IG) correctly models the feasible set of the given implication graph G.*

Proof Let D be the decision diagram generated by (IG), $\mathrm{Sol}(D)$ be the set of solutions represented by s-t paths in D, and $\mathrm{Sol}(G)$ be the feasible set of the implication graph G. We want to show that $\mathrm{Sol}(D) = \mathrm{Sol}(G)$.

The implication constraints of G enforce that if x_j is set to v_j, then x_k must be set to v_k for all nodes $x_k^{1-v_k}$ that are reachable from $x_j^{v_j}$. Since the transition function only enforces these constraints, it cannot eliminate feasible solutions. This implies that all feasible solutions must be represented as s-t paths in D. Therefore, $\mathrm{Sol}(G) \subseteq \mathrm{Sol}(D)$.

To show that $\mathrm{Sol}(D) \subseteq \mathrm{Sol}(G)$, let $\hat{x}$ be a solution represented by an s-t path in D. We want to show that $\hat{x}$ is feasible with respect to G. Suppose for contradiction that $\hat{x}$ is infeasible. Then $\hat{x}$ must violate the constraint of some arc $(x_j^{v_j}, x_k^{v_k})$ in G. That is, $\hat{x}_j = v_j$ and $\hat{x}_k = 1 - v_k$. Assume without loss of generality that x_j comes before x_k in the ordering, which can be done because each arc $(x_j^{v_j}, x_k^{v_k})$ in G has a counterpart $(x_k^{1-v_k}, x_j^{1-v_j})$. Then this assignment cannot occur because $t_j(s^j, v_j)$ enforces the domain of x_k to become $\{v_k\}$ in all subsequent states, which is a contradiction. Therefore, $\hat{x}$ is feasible, and thus $\mathrm{Sol}(D) = \mathrm{Sol}(G)$. □

In fact, formulation (IG) is correct even in a depth-d variant in which we redefine $\bar{R}_{k,j,v_j}$ in the transition function to only consider nodes within a distance of d from $x_j^{v_j}$, for any $d \geq 1$. Note that the exact same proof above holds in this case. This variant may be useful to improve the performance of equivalence tests, at the cost of potentially allowing some equivalent nodes to be left unmerged.

4.1.2 Completeness

Now that the correctness of (IG) is established, we turn to the question of when this formulation yields a reduced decision diagram. It suffices to present an efficient complete equivalence test – that is, a test that identifies exactly when two states have the same completion set.

The example in Figure 3 shows that (IG) as currently formulated does not always generate a reduced decision diagram. In Figure 3a, the two second-layer nodes have the same completion set but different states. The state of the leftmost second-layer node unnecessarily has 0 in the domain of x_3, which if removed, would enable merging. This observation motivates the lemma below, which provides a sufficient condition for completeness.

We call a state s domain consistent if for every variable x_i and value $v_i \in \mathcal{D}_s(x_i)$, there exists a feasible completion from state s that assigns v_i to x_i. We use a natural equivalence test in formulation (IG), which simply identifies two nodes as equivalent if they have the same state.

Lemma 1 *If every state in (IG) is domain consistent, then the equivalence test from formulation (IG) is complete.*

Proof Domain consistency ensures that if two domains are different, then they must have different completion sets. More formally, consider states s_1 and s_2 with different domains. Suppose without loss of generality that there exists $v_i \in \mathcal{D}_{s_1}(x_i)$ such that $v_i \notin \mathcal{D}_{s_2}(x_i)$. Then the above property implies there exists a completion from s_1 which assigns v_i to x_i that does not exist from s_2. □

As a side note, observe that Lemma 1 holds not only for the formulation (IG), but also for any formulation in which the state space consists of domains for their completions.

The next step is to provide a means to obtain domain consistency at every state, since this would yield completeness. In fact, it turns out that the transition function $t_j(s^j, v_j)$ in (IG) preserves domain consistency as long as the original state s^j is also domain consistent, as we establish next with Theorem 1. This implies that it is sufficient to make the root state domain consistent, which can be done in linear time in the size of the conflict graph as shown later in this section. For instance, in Figure 3 it would suffice to make the initial state domain consistent in order to construct a reduced decision diagram.

Theorem 1 *If s is a domain consistent state, then $t_j(s, v_j)$ is a domain consistent state if feasible.*

In order to prove Theorem 1, we first derive two intermediate lemmas. We use the following theorem from Aspvall et al. [6], which characterizes feasibility of an implication graph.

Theorem 2 (Aspvall et al. [6]) *An implication graph G is feasible if and only if there is no x_j such that x_j^0 and x_j^1 are in the same strongly connected component.*

The two intermediate lemmas are the following.

Lemma 2 *Given an implication graph G, there exists a feasible solution with x_j set to v_j if and only if there exists a feasible solution for the implication graph $\hat{G} := G \cup \{(x_j^{1-v_j}, x_j^{v_j})\}$.*

Proof The constraint from the additional arc $(x_j^{1-v_j}, x_j^{v_j})$ is violated exclusively by all solutions with x_j set to $1 - v_j$, leaving exactly the feasible solutions of G with x_j set to v_j. □

Lemma 3 *Given a feasible implication graph G, there exists a feasible solution with x_j set to v_j if and only if there is no path from $x_j^{v_j}$ to $x_j^{1-v_j}$ in G.*

Proof By Lemma 2, there is a feasible solution with x_j set to v_j if and only if $\hat{G} := G \cup \{(x_j^{1-v_j}, x_j^{v_j})\}$ is feasible. In view of Theorem 2 and the feasibility of G, $\hat{G}$ is feasible if and only if adding $(x_j^{1-v_j}, x_j^{v_j})$ to G keeps $x_j^{1-v_j}$ and $x_j^{v_j}$ in different strongly connected components.[1] This happens if and only if there is no path from $x_j^{v_j}$ to $x_j^{1-v_j}$ in G. $\square$

Lemma 3 tells us how to achieve domain consistency for the implication graph. For every variable x_j, we check if x_j^0 is reachable from x_j^1 and vice versa. If $x_j^{1-v_j}$ is reachable from $x_j^{v_j}$, we remove v_j from $\mathcal{D}_s(x_j)$.

In addition, this can be done in linear time as follows. Tarjan's strongly connected components algorithm [35] provides the strongly connected components in reverse topological order. By treating each component as a node, we can scan the graph in topological order in a single pass to find these paths. Throughout this pass, we store at each component the variable-value assignments of its ancestors in order to pass it forward. Whenever we find the complement of one of these assignments, we can remove the assignment from the domain.

We now prove Theorem 1, which implies that it suffices to ensure domain consistency at the root state in order to guarantee completeness.

Proof (Theorem 1) Consider the states s and $s' := t_j(s, v_j)$. In order to show that s' is domain consistent, we need to show that for any $v_k \in \mathcal{D}_{s'}(x_k)$, there exists a completion from s' that assigns v_k to x_k.

Let $\hat{G}_s$ be the implication graph G with the additional arcs $(x_j^{v_j}, x_j^{1-v_j})$ for all $v_j \in \{0,1\} \setminus \mathcal{D}_s(x_j)$. Note that the feasible set of $\hat{G}_s$ corresponds to the completion set of s by Lemma 2. Moreover, the feasible set of $G' := \hat{G}_s \cup \{(x_j^{1-v_j}, x_j^{v_j})\}$ corresponds to the completion set of s'. Following Lemma 3, it suffices to show that G' does not contain a path from $x_k^{v_k}$ to $x_k^{1-v_k}$.

Given that $\mathcal{D}_s$ is consistent, there must exist a completion x from s such that $x_k = v_k$. Equivalently, $\hat{G}_s$ must not contain a path from $x_k^{v_k}$ to $x_k^{1-v_k}$ according to Lemma 3. Therefore, any path in G' from $x_k^{v_k}$ to $x_k^{1-v_k}$ must go through the only new arc $(x_j^{1-v_j}, x_j^{v_j})$. However, $x_k^{1-v_k}$ is not reachable from $x_j^{v_j}$, as otherwise v_k would be removed from $\mathcal{D}_{s'}(x_k)$ as a result of the transition function. Hence, there cannot be a path in G' from $x_k^{v_k}$ to $x_k^{1-v_k}$. $\square$

The above theorem directly implies the following result.

Corollary 1 *The equivalence test from the DP formulation (IG) is complete when the initial state is domain consistent.*

[1] As a technicality, this requires that Theorem 2 holds when there are arcs between the two nodes of a same variable. Despite assuming a standard implication graph, the proof from Aspvall et al. [6] is also valid with same-variable arcs.

Therefore, once we establish domain consistency in the root state, we can use the DP formulation in a top-down fashion to construct a reduced decision diagram.

We remark that this serves as an alternative proof for complete equivalence for the independent set problem in [17]. If we view the independent set problem in terms of an implication graph, we obtain a graph where all arcs point from a nonnegated node to a negated node. This means that every path in the implication graph has length at most one, and thus it suffices for the transition function to consider only the neighbors of each vertex, as done in the formulation of [17]. Moreover, the initial domain of all possibilities (i.e. the root state of the formulation in [17]) is always consistent since every individual vertex of the original graph is a feasible independent set.

4.2 Variable ordering

Variable ordering for decision diagrams is often based on heuristics. Using a fast heuristic is particularly helpful in our case, as we may be generating several decision diagrams during the solution process of a single problem.

Based on the close connection of conflict graph constraints to independent set constraints, we use a generalization of a variable ordering heuristic for independent set that has shown to work well in practice, namely the minimum number of states ordering [18, 17]. In the context of independent set, at each layer, the ordering selects the vertex v that appears in the fewest number of states in the state pool. Every node with a state in which v appears will branch to both zero and one, whereas if v does not appear, the corresponding node only branches to zero. Therefore, this minimizes the number of arcs in the following layer.

A natural generalization for conflict graph constraints is as follows: at each layer, we select the variable with the smallest sum of domain sizes throughout the state pool. This minimizes the number of arcs in the next layer since each assignment corresponds to an arc, given that the domains are consistent. We use this ordering in our computational experiments discussed in Section 7.

5 Generic Constraints

Focusing on a substructure is typically only practical if it captures most of the problem, and any constraints not part of the substructure are still taken into account in some form.

Suppose that our goal is to solve $\max_x\{c^\top x : Ax \leq b,\ x \in \{0,1\}^n\}$, given an objective function $c \in \mathbb{R}^n$, a coefficient matrix $A \in \mathbb{R}^{m\times n}$, and a right-hand side $b \in \mathbb{R}^m$. While we focus on the binary version, in general we can replace the binary constraints by bounded integer constraints. We partition the constraints $Ax \leq b$ into two sets, given by $\hat{A}x \leq \hat{b}$ and $\bar{A}x \leq \bar{b}$, where the latter set of constraints will be represented with a (relaxed) decision diagram. We denote the remaining constraints $\hat{A}x \leq \hat{b}$ by *generic constraints*.

As mentioned in the introduction of this paper, we aim to solve the following relaxation of the above problem:

$$\max_x \{c^\top x : \hat{A}x \leq \hat{b},\ x \in \text{conv}(S)\}, \tag{1}$$

where S is a superset of $\{x \in \{0,1\}^n : \bar{A}x \leq \bar{b}\}$. Here, S is represented by a decision diagram.

The partition of the constraints may be induced by a chosen substructure. More precisely, we first construct a relaxed decision diagram for a substructure of the problem, representing S, and then mark as generic the constraints that are not made redundant by the constraint $x \in \text{conv}(S)$. In this case, identifying generic constraints can be done by checking if at least one solution represented in the decision diagram is violated by the constraint. In other words, a constraint $a^\top x \leq b$ can be marked as generic if $\max_{x \in S}\{a^\top x\} > b$, which can be efficiently checked by finding a maximum weight path in the decision diagram. All other constraints can be discarded, as they are implicitly represented in the decision diagram.

In the context of conflict graphs, we mark a constraint as generic in our implementation if it does not have a particular form implied by conflict constraints: $\sum_{i \in P} x_i + \sum_{i \in N}(1 - x_i) \leq 1$ for some disjoint set of variable indices P and N. Although this can be checked quickly, it is possible that we label more constraints than necessary as generic. Note that in this particular implementation, we may also remove constraints that are not completely redundant with respect to $\text{conv}(S)$ when S corresponds to a relaxed decision diagram.

We handle generic constraints in two ways: Lagrangian relaxation and constraint propagation. In Lagrangian relaxation, we essentially seek to solve (1). In constraint propagation, we strengthen the set S by removing some of the solutions violated by $\hat{A}x \leq b$.

5.1 Lagrangian relaxation

Lagrangian relaxation is a classical technique that is primarily used to obtain dual bounds for optimization problems. It consists of moving a set of constraints to the objective function by penalizing its violation. In our context, we apply Lagrangian relaxation with respect to the generic constraints, by solving the following optimization problem:

$$\min_{\lambda \geq 0} \max_x \{c^\top x + \lambda^\top(\hat{b} - \hat{A}x) : x \in \text{conv}(S)\}.$$

The variables λ are called Lagrange multipliers, which represent penalties for the violation of the constraints $\hat{A}x \leq \hat{b}$.

This problem can be solved with subgradient methods that require optimizing a linear function over $\text{conv}(S)$ as a subproblem. In our context, this subproblem entails finding an optimal path in the decision diagram representing S, which can be done in linear time with respect to the size of the decision diagram, once it is constructed. This makes decision diagrams particularly well-suited to be used in conjunction with Lagrangian relaxation, as often several of these subproblems need to be solved. The use of Lagrangian relaxation with decision diagrams has been previously investigated in the context of constraint programming as well [14, 13].

Lagrangian relaxation theory establishes that the optimal value of the above problem is equivalent to the optimal value of (1). This provides a clean interpretation of the bound we obtain from Lagrangian relaxation. Essentially, we are optimizing over the convex hull of the set of points represented by the decision diagram intersected with the generic constraints in their original linear form. In other words, we are convexifying the constraints involved in the construction of the decision diagram, taking integrality into account.

We remark that the above problem can also be solved by modeling $\mathrm{conv}(S)$ as a network flow with additional arc variables [9, 14, 12, 36] and solving the overall linear program. However, we opt for the Lagrangian relaxation approach as it tends to produce good bounds quickly in our computational experience, interrupting it before reaching optimality.

A limitation of Lagrangian relaxation is that it is only equivalent to adding the constraints back in its original linear form. In some cases, we may need to tighten these generic constraints in order to obtain improvements. For instance, if the decision diagram is constructed from a set of linear constraints whose polyhedron has only integer vertices, then this approach cannot yield a better bound than the LP bound.

5.2 Constraint propagation

Even if a generic linear constraint results in a large decision diagram by itself, it can be partially incorporated into the decision diagram of other constraints without significantly increasing its size. This is particularly true if we use domain states: we use constraint propagation to filter out infeasible values from the domain states [5, 27]. This results in the elimination of infeasible points from the decision diagram, which may improve the associated bounds. Moreover, it may reduce the time it takes to construct the decision diagram, as we are potentially exploring fewer nodes.

Consider a constraint $a^\top x \leq b$ and a node u with domain state s. Given a variable x_j and a value v_j in the domain $\mathcal{D}_s(x_j)$, our goal is to determine before branching on u if no completion assigning v_j to x_j satisfies the constraint. If so, we can remove v_j from the domain $\mathcal{D}_s(x_j)$.

Before approaching this problem, let us consider an easier variant. Suppose that we want to tackle this problem on a fully constructed decision diagram. This is equivalent to determining if the constraint is violated by all possible solutions with $x_j = v_j$ corresponding to paths that pass through the node u. To solve this, we can find the smallest left-hand side $a^\top x$ within this solution set and check if it exceeds the right-hand side b. This fundamental propagation idea is extensively used in constraint programming and MIP solvers [24, 10, 2, 3, 34]. In its simplest version, variable bounds are used to minimize $a^\top x$, but here we express this minimization problem in terms of the partial solution set and the completion set of the node u.

Denote by $S^{\downarrow}(u)$ the partial solution set of a node u at layer k: the set of all solutions $(x_1, \ldots, x_{k-1})$ corresponding to a path from the root to u. Similarly, denote by $S^{\uparrow}(u)$ the completion set of u, the set of all solutions $(x_k, \ldots, x_n)$ corresponding to a path from u to the terminal node.

Proposition 2 *Consider a decision diagram D ordered $x_1, \ldots, x_n$, a node u of D at layer k, and a linear constraint $\sum_{i=1}^{n} a_i x_i \leq b$. Let $j \geq k$. Then no solution x with $x_j = v_j$ corresponding to paths in D containing u satisfies the linear constraint if and only if*

$$\min_{(x_1,\ldots,x_{k-1}) \in S^{\downarrow}(u)} \left\{ \sum_{i=1}^{k-1} a_i x_i \right\} + \min_{\substack{(x_k,\ldots,x_n) \in S^{\uparrow}(u) \\ x_j = v_j}} \left\{ \sum_{i=k}^{n} a_i x_i \right\} > b$$

Proof The set of solutions x corresponding to paths containing u is $S(u) := \{x : (x_1, \ldots, x_{k-1}) \in S^{\downarrow}(u), (x_k, \ldots, x_n) \in S^{\uparrow}(u)\}$. Let $S'(u) := S(u) \cap \{x : x_j = v_j\}$, which is the set of solutions for which we want to check violation. This set violates the constraint if and only if $\min_{x \in S'(u)} \sum_{i=1}^{n} a_i x_i > b$, which is equivalent to the above condition. $\square$

For convenience, we denote the first minimization term on the left-hand side of the condition in Proposition 2 by $p_a(u)$ and the second term by $c_a(u, j, v_j)$.

Let us return to the context of filtering the domain of a node u at the construction stage. We can efficiently compute $p_a(u)$, since it consists of optimizing a linear function over the decision diagram of the partial solution set of u, which is fully available at the time of branching for a top-down construction. It is not necessary to recompute $p_a(u)$ at every node, as we can maintain them throughout the construction. At every new node u' coming from a node u and arc $x_j = v_j$, we let $p_a(u') := p_a(u) + a_j v_j$. In addition, whenever two nodes u and u' are merged into u'', we let $p_a(u'') = \min\{p_a(u), p_a(u')\}$.

On the other hand, computing $c_a(u, j, v_j)$ during construction is difficult because we do not have the completion set of the node. Instead, we compute a lower bound B for $c_a(u, j, v_j)$. If we satisfy the condition $p_a(u) + B > b$, then by Proposition 2 we can still safely remove v_j from $\mathcal{D}_s(x_j)$. In our case where domains are states, we calculate B by minimizing $\sum_{i=k}^{n} a_i x_i$ over the possible values of the domains, after restricting x_j to be v_j. More precisely, we let B be $\sum_{i: a_i \geq 0, i \neq j} \min(\mathcal{D}_s(x_i)) + \sum_{i: a_i < 0, i \neq j} \max(\mathcal{D}_s(x_i)) + a_j v_j$.

This completes the description of the constraint propagation method. We remark that while this approach can only improve the bound since it removes infeasible solutions, it can potentially increase the size of the decision diagram. A simple example where this happens is given in Figure 4. Alternatively, if we want to ensure that the size of the decision diagram does not increase, we may

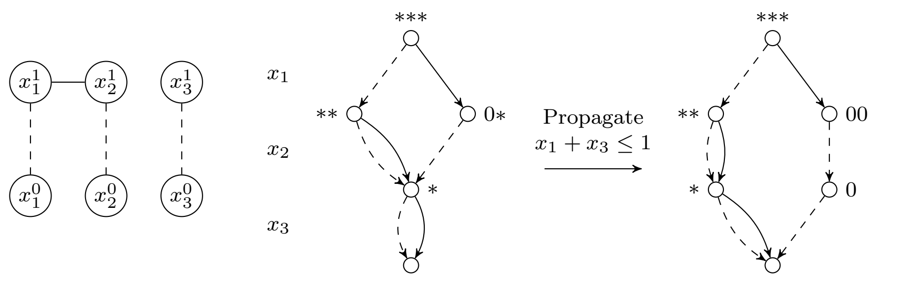


Fig. 4: An example in which propagation increases the size of the decision diagram in the context of conflict graphs.

apply propagation only with respect to the variable of the next layer. The size cannot increase because no states are modified with this approach, except for identifying infeasible nodes to be pruned. This can be interpreted as a weaker version of arc filtering [5], which we call *arc pruning.*

Finally, we remark that one could detect special classes of constraints and implement specialized propagators. Our implementation is however limited to this more general propagator.

6 Primal bounds

In our framework, primal bounds can not only be generated from decision diagrams for the MIP solver, but conversely primal bounds from the MIP solver can also benefit decision diagram construction.

We next present heuristic approaches to identify feasible solutions from the relaxed decision diagrams we use. We consider two cases: one when generic constraints are not present and one when they are. In the latter case, we assume that we use Lagrangian relaxation as described in Section 5.1.

1. **Without generic constraints.** During the construction of a relaxed decision diagram, we keep track of nodes that have been merged due to relaxation. We then find the optimal path that does not contain any of such nodes. This path corresponds to a feasible solution to the overall problem because it only contains exact nodes.
2. **With generic constraints.** The process of solving the Lagrangian relaxation problem typically involves optimizing over the decision diagram a number of times. The solutions obtained in this process are called primal iterates. For every such solution generated, we check its feasibility with respect to the overall problem. If we find that it is feasible, we store it as a primal feasible solution. This simple approach has been suggested in early works on Lagrangian relaxation [26].

An alternative is to generate primal feasible solutions (and thus primal bounds) from restricted decision diagrams [19], which encode a subset of feasible solutions. However, we opt not to investigate this approach, as not only constructing further decision diagrams for primal bounds can be inefficient, but also they require all constraints to be considered in the construction.

Conversely, primal bounds from the MIP solver can help eliminate solutions from the decision diagrams, potentially making them smaller. If we have a primal bound B_p and we are maximizing an objective c, then we can effectively add the constraint $c^\top x \geq B_p$ to the decision diagrams. In order to keep the size of the decision diagram in check, we do so by applying arc pruning with respect to this constraint, as described in the end of Section 5.2.

7 Computational experiments

Given that we focus on the conflict graph substructure, we first need to understand the impact of the DD bounds in a form where the problem is entirely composed of that structure. In particular, we first run experiments on a pure independent set problem, which can be represented by a conflict graph. We then investigate the impact of the DD bounds in the presence of side constraints by adding knapsack constraints to an independent set problem. As a MIP solver, we use SCIP 5.0.1 equipped with CPLEX 12.6 as an LP solver. SCIP was chosen in part because it enables us to directly access the conflict graph. The experiments were performed on a 2.33Ghz Linux machine with 32GB of RAM. The code can be found at `https://github.com/ctjandra/ddopt-bounds`.

As discussed in Section 3, we generate bounds at certain nodes of the branch-and-bound tree, passed to SCIP via its relaxation handler. The bound is computed after solving each LP and may be done more than once per node if it involves multiple LPs. The main input to the SCIP relaxator is the conflict graph (in the form of a clique table), which is built by SCIP after the presolve step. Any additional constraints for propagation and Lagrangian relaxation are copied over from the first LP at the root, after presolve. As a result, cutting planes are not considered for bound generation. In addition, due to a technical incompatibility, we disable restarts and variable aggregation for all runs.

To keep the experiments clean, we opt for the following simple approaches in our implementation. Improving upon these is left for future work.

1. To select the nodes at which a bound will be generated, we use a simple node selection rule: we generate bounds only when the number of variables of the subproblem is below a given threshold. This is motivated by the computational observation that the bounds are more likely to help for smaller subproblems in this particular experimental setup.
2. We build a new decision diagram from scratch at every node we generate a bound. Besides simplifying the implementation, this avoids potential memory concerns from handling more than one decision diagram at a time.

7.1 Independent set constraints

We consider random graphs parameterized by size n and density d following the Erdős–Rényi model $G(n, d)$: each edge of a graph with n vertices is included with probability d. We also consider instances from the DIMACS maximum clique benchmark set [25], discussed later.

For the random graphs, we select two instance sizes n, 150 and 300, and vary the density d parameter from 10% to 90% in increments of 10%. For each of these parameters, we generate 16 instances. All solving times and number of nodes reported are shifted geometric means among these 16 instances with a shift factor of 10 for solving time and 100 for nodes. We set a time limit of one hour.

Except when stated otherwise, we generate bounds for every subproblem with at most 2/3 of the variables – that is, 100 and 200 for the instances of sizes 150 and 300 respectively. This subproblem size is manually tuned: we performed some computational experiments with different sizes and selected one that performed well for these runs.

The IP model given to the solver is a clique cover formulation. The constraints are $\sum_{j \in C} x_j \leq 1$ for every clique C in a clique cover $\mathcal{C}$, which is a set of cliques that covers all edges of G. Each clique is generated by starting with a vertex with maximum degree and greedily adding vertices with maximum degree that form a clique with the current set.

In practice, we would typically not use a MIP solver to solve the maximum independent set problem, as this is a well-studied problem with several specialized algorithms significantly faster than a MIP solver. However, they often no longer function as is when the problem is further constrained, as is often the case with real-world problems. Observing the behavior of a MIP solver in this simpler case is useful as a stepping stone to problems with independent set constraints as a substructure, which is examined in the next set of experiments in Section 7.2.

In terms of implementation, we construct decision diagrams based on the conflict graph of the problem, following Section 4, even though we know these are independent set instances. Since the DP formulation of the conflict graph generalizes the formulation for the independent set problem, the resulting decision diagrams are the same, presolve aside. The difference between this version and one specific to the independent set problem is overhead in time from extracting and processing conflict constraints. Note also that presolve may affect these graphs including their densities. Any figure in this section refers to the original densities.

We use arc pruning with primal bounds and the primal heuristic without generic constraints as described in Section 6. We use the variable ordering described in Section 4.2, and a merging rule that merges together nodes with the smallest objective values until the width limit is satisfied.

- **Overall performance:** The first plots in Figures 5 and 6 show the overall performance of the method. For random graphs, solving maximum inde-

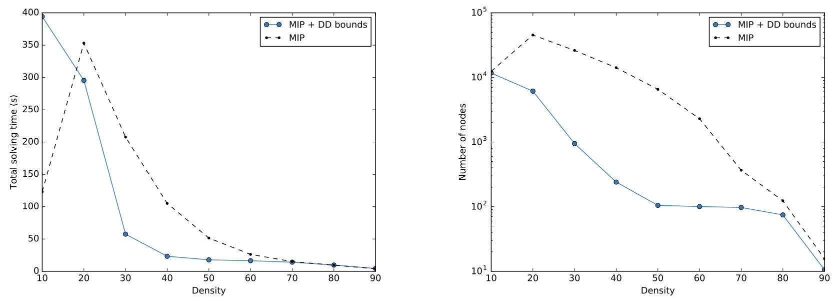


Fig. 5: Comparison in solving time and branch-and-bound tree size between applying and not applying DD bounds in independent set instances with 150 vertices. Results are averaged over solving 16 instances to optimality for each density parameter.

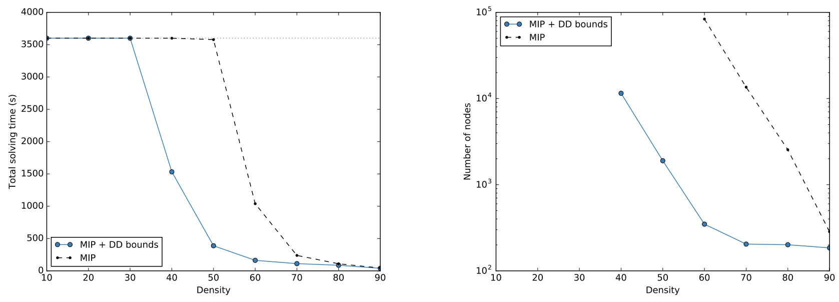


Fig. 6: Comparison in solving time and branch-and-bound tree size between applying and not applying DD bounds in independent set instances with 300 vertices. The gray dotted line on the left plot indicates the time limit of one hour. Unreported data on number of nodes corresponds to cases in which the time limit of one hour was hit on the majority of runs.

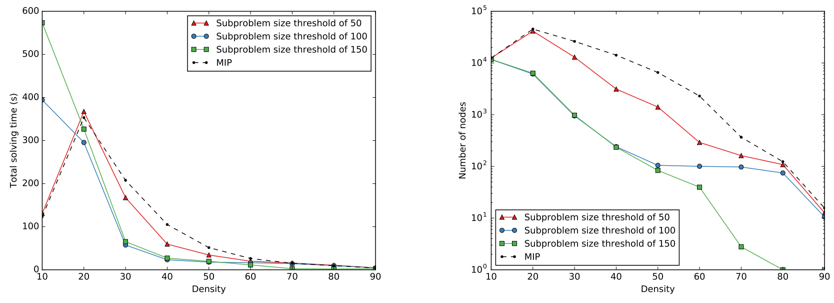


Fig. 7: Comparison of different subproblem size thresholds with 150 vertices.

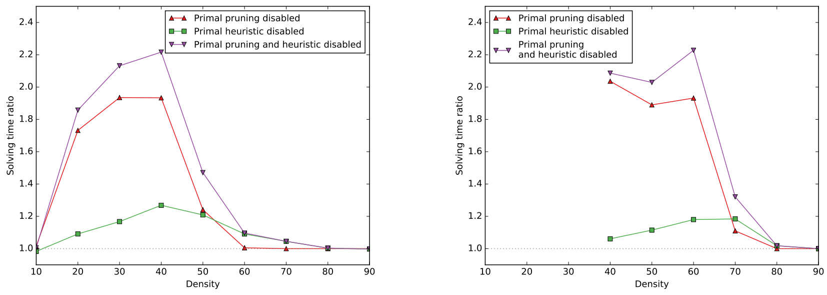


Fig. 8: Effect on solving time when disabling primal pruning and/or primal heuristic on instances of size 150 (left) and 300 (right). The vertical axis is the ratio t_d/T, where t_d is the time without a given feature and T is the time with all capabilities, and the horizontal axis is the density parameter of the set of instances. On the right, data points for densities below or equal 30 are omitted because the time limit is always hit.

pendent set problems tends to be easier for either dense graphs or very sparse graphs, and this is evident from the plot. The graph suggests that our approach is more beneficial around middle ranges of density. For instance, in the case of density 60% and 300 vertices (in which time limit is not hit), applying the DD bounds reduces the tree size by 99.6%, allowing the problem to be solved 6.4 times faster. The bounds do not perform well for low densities, consistent with observations from previous works [17, 36].

– **Subproblem size:** Figure 7 shows what happens if we choose a different subproblem size threshold. Selecting a larger threshold – that is, applying decision diagram bounds more often – can be helpful when we know relaxed decision diagrams are strong, such as with high-density cases. However, that can waste time when the relaxations do not scale well, as illustrated by the low-density cases, in which case focusing on smaller subproblems performs better.
  We omit the plot for instances of size 300 as it depicts a similar behavior as above (although it cannot be observed for lower densities due to the time limit).
– **Primal bounds:** Figure 8 illustrates the impact of primal bounds. It provides the ratio between the solving time without a given feature (primal pruning and/or primal heuristic) and the solving time with all features. The effect of the primal techniques becomes more significant with larger branch-and-bound trees. Primal pruning is particularly helpful to avoid wasting time on small nodes that are easily identifiable as infeasible.

In Section A of the appendix, we provide further information on the time spent generating bounds and on the number of times an improving bound was found.

Table 1 exhibits solving times for instances from the DIMACS maximum clique benchmark set [25], converted to maximum independent set by considering their complement graphs. Most of these instances have different structures than random Erdős–Rényi graphs. We present only the set of instances that were solved to optimality within one hour with SCIP 5.0.1 at its default settings, either with or without bounds from decision diagrams, and for at least one random seed. The bounds from decision diagrams are applied at every branch-and-bound node with at most 3/4 of the total number of vertices with a width of 100.

On average across all DIMACS instances tested, we observe that using the bounds makes the solving process 1.87x faster with a node reduction of 86.7%. In practice, we would not apply this method to low-density instances however. If we consider only instances with density at least 30%, the improvement is more pronounced: the solving times are on average 3.29x faster and the node reduction is on average 94.8%.

7.2 Independent set and knapsack constraints

We next wish to understand the impact of the DD bounds in presence of side constraints. The instances we consider in this section are a combination of independent set (set packing) constraints with knapsack constraints. The integer programming model is given by:

$$\begin{aligned}
\max\ & c^\top x \\
& \sum_{j \in C} x_j \le 1 && \text{for all } C \in \mathcal{C} && \text{(set packing)} \\
& \sum_{j=1}^{n} a_{ij} x_j \le b_i && \text{for all } i = 1, \dots, m_{\text{knap}} && \text{(knapsack)} \\
& x \in \{0,1\}^n
\end{aligned}$$

The independent set constraints for the input graph G are modeled with a clique cover formulation, as described in the previous section.

To eliminate the bias from the properties of the conflict graph that we studied in the previous section, we define an underlying random graph structure for which the performance of the decision diagram is rather stable: the Watts–Strogatz model [37], which has small-world properties and allows us to scale up to relatively large size without compromising the quality of the DD performance too much. This model generates graphs through the following process. Given the desired number of vertices n, the desired mean degree k (assumed even), and a probability p, construct a preliminary graph with n vertices arranged in a cycle. Let two vertices be adjacent if and only if they are within distance $k/2$ in the cycle. Then for each vertex i and outgoing edge (i, j), reassign j with probability p to another vertex (besides i or a neighbor

		Solving time (s)		Number of nodes	
Instance	Density	MIP	MIP + DD	MIP	MIP + DD
`brock200_1`	25.5%	>3600.00	1096.36	>239552.41	14637.51
`brock200_2`	50.4%	245.74	39.59	19631.05	173.37
`brock200_3`	39.5%	844.77	82.23	57032.38	625.98
`brock200_4`	34.2%	1321.67	166.97	93581.85	1583.24
`C125.9`	10.2%	21.47	45.49	1370.75	1119.28
`gen200_p0.9_44`	10.0%	32.88	105.87	737.29	1149.75
`gen400_p0.9_65`	10.0%	>2366.88	>3600.00	>58268.84	>11993.40
`gen400_p0.9_75`	10.0%	995.33	1742.15	6968.81	4035.27
`hamming8-4`	36.1%	142.47	78.00	1672.93	179.32
`keller4`	35.1%	288.55	32.99	30392.73	357.44
`MANN_a27`	1.0%	42.79	34.72	5006.48	4146.73
`MANN_a45`	0.4%	797.68	820.63	88823.52	79162.73
`p_hat300-1`	75.6%	195.28	96.01	7684.60	151.50
`p_hat300-2`	51.1%	1184.74	459.30	26900.27	2355.57
`p_hat500-1`	74.7%	2688.06	700.69	79198.67	609.70
`san200_0.7_2`	30.0%	22.39	16.74	60.25	31.80
`san200_0.9_3`	10.0%	13.68	56.64	239.43	553.39
`san400_0.5_1`	50.0%	495.26	87.67	320.56	32.58
`san400_0.7_1`	30.0%	490.55	392.30	1152.63	379.04
`san400_0.7_2`	30.0%	885.74	380.64	6448.98	651.72
`san400_0.7_3`	30.0%	>3439.64	>1431.74	>51602.73	>4618.31
`san400_0.9_1`	10.0%	254.77	419.46	470.31	709.16
`sanr200_0.7`	30.3%	2973.10	407.97	270050.01	4891.10
`sanr400_0.5`	49.9%	>3600.0	2544.89	>133516.18	8338.75
Average	all	448.47	239.15	9920.72	1322.03
	$\geq$ 30%	733.01	223.05	16145.26	835.61

Table 1: Effect of applying bounds on selected DIMACS benchmark instances. Averages are in shifted geometric mean. Entries marked with a '>' have reached the time limit for at least one random seed.

of i) uniformly chosen at random. In our instances, the mean degree k is 100 and the probability p is 0.1.

We perform two sets of experiments: we first assess the quality of the DD bounds on a varying number of knapsack constraints, and then their impact on the overall solving process for a fixed fraction of knapsack constraints. We use the following parameters for both cases. For each knapsack constraint indexed by i, we select a support of 100 variables at random and choose coefficients a_{ij} uniformly at random from 1 to 100, and the remaining variables have zero coefficients. We maximize an objective with coefficients c_j also randomly chosen from 1 to 100. We fix b_i to 150 in all instances in this section. Solving times and nodes are aggregated over 10 instances with 5 solver random seeds each using shifted geometric mean with a shift of 10 for time and 100 for nodes.

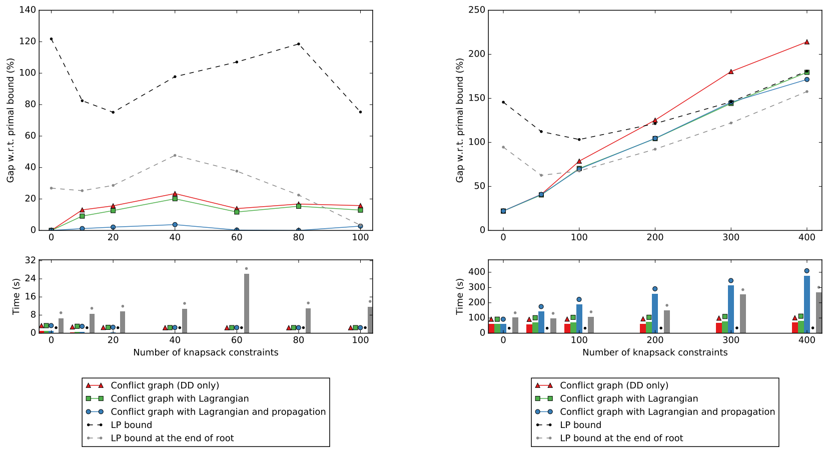


Fig. 9: The plot on the top illustrates a comparison of bound quality for independent set + knapsack constraints with 200 (left) and 1000 (right) variables, varying the number of knapsack constraints. The plot on the bottom represents the solve times to reach each bound. The quality of a bound D is represented by its gap $(D - P)/P$, where for $n = 200$ (left), P is the optimal value, and for $n = 1000$ (right), P is the best primal value given by the solver at its default settings after 10 minutes of solve time.

The variable ordering and merging rules for the relaxed decision diagrams are the same as in the previous section. We use different width limits for the two sets of experiments: 1000 for the first set and 100 for the second one. We observed that increasing the latter to 1000 results in similar behavior, with little improvement to pruning, likely because we only focus on small subproblems that do not require large widths to be effectively tackled. The Lagrangian relaxation is solved using the ConicBundle library, which implements a bundle method to solve it. We extract the best bound it finds by the end of 50 iterations. We do not use warm starts for the Lagrangian relaxation.

In the first set of experiments, we are interested only in examining the bounds at the root node. Figure 9 shows the quality of the bounds (represented by gap, as defined in the caption of the figure) as we increase the number of knapsack constraints in the cases of $n = 200$ and $n = 1000$. As a baseline, we include in the plots the initial LP bound and the LP bound at the end of the root node, which may include cutting planes, at default settings. In addition, the figure provides the time it takes to reach each bound.

We observe that for small instances, the bounds are strong even if several knapsack constraints are present. For the larger instances, more knapsack constraints result in lower DD bound quality. This is expected since we mostly explore the conflict graph substructure. We also show the DD bounds with constraint propagation disabled, and both constraint propagation and Lagrangian relaxation disabled. Enabling both of them is particularly helpful for the smaller instances, but for larger instances, we observe that Lagrangian relaxation suffices to improve the bound most of the way at a relatively small cost. Moreover, Lagrangian relaxation and constraint propagation are more helpful when more knapsack constraints are present, which is expected given that their role is to take them into account.

Our final set of experiments assesses the impact of the DD bound on the overall search tree. Again, we follow the Watts–Strogatz graph model to generate instances with n variables, and we vary the number of variables n from 300 to 450 in increments of 50. We keep the number of knapsack constraints at a fixed proportion to n. Experiments are performed for 16 random instances, each with 5 different MIP solver random seeds. For this set of experiments, we do not apply a time limit. Based on the results presented in Figure 9, we determine that adding $0.1n$ knapsack contraints strikes a good balance between the relative performance of the LP bound and the DD bound for an insightful comparison. The subproblem size threshold is 100, determined by manual tuning. We use Lagrangian relaxation and constraint propagation as described in Section 5. Moreover, we include the primal heuristic (with generic constraints) and the arc pruning based on primal bounds described in Section 6.

- **Overall performance:** The overall performance of the decision diagram bounds is presented in Figure 10, along with a summary in relative terms in Table 2. On average for these instances, this technique results in an overall speed-up of 66.94% (or equivalently, a slowdown of 40.09% if we disable the bounds). The number of nodes is reduced by 66.10% – to almost one-third of its original size. From Table 2, we observe that the speed-up scales well up to the sizes we tested. Figure 11 illustrates all individual instances and random seeds and it suggests that the approach is fairly robust for these instances.
- **Lagrangian relaxation, constraint propagation, primal bounds:** Table 3 shows the effect of disabling each of the techniques we use on top of the dual bound generation. While the dual bounds by themselves are strong – in part because the independent set constraints play a substantial role in defining the problem – removing from consideration either the generic constraints or the primal techniques results in a significant deterioration of the speed-up. In particular, although removing one of the two generic constraint techniques does not affect the solving time too much, disabling both of them has a large impact, indicating that it is important to consider the generic constraints in some way.

Section A of the appendix contains information on the fraction of time spent generating bounds and on the number of times the LP bound was improved.

7.3 Computational limitations

The previous subsections provide computational evidence that our approach works well in scenarios where the conflict graph either forms the entire prob-

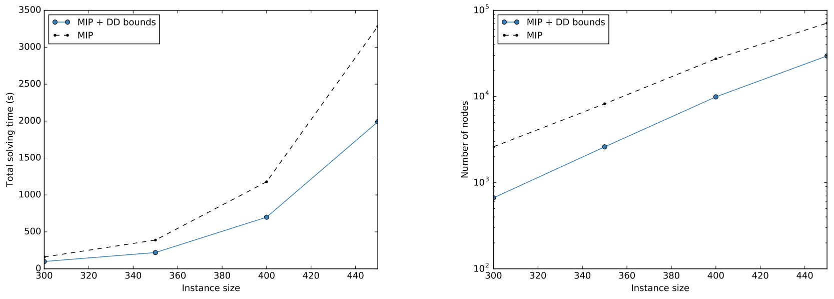


Fig. 10: Comparison in solving time and branch-and-bound tree size of the use of decision diagram bounds for independent set + knapsack instances.

	Instance size			
	300	350	400	450
Speed-up (%)	62.13	76.78	68.57	64.90
Node reduction (%)	74.46	68.34	63.84	58.40

Table 2: Speed-up and node reduction from using decision diagram bounds. Speed-up is the ratio of original solving time to the solving time with the bounds, minus one (e.g. a speed-up of 100% means twice as fast). Solving times and numbers of nodes are averaged using the shifted geometric mean (with shifts of 10 and 100 respectively).

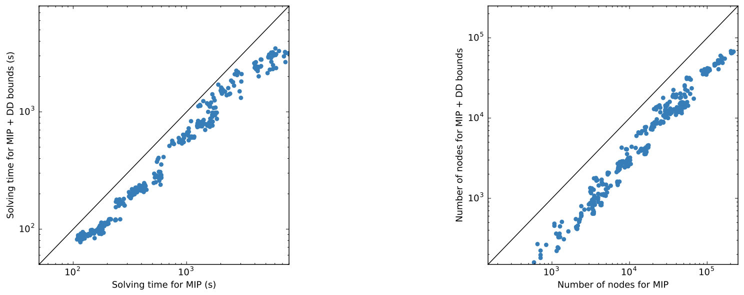


Fig. 11: Effect of using decision diagram bounds illustrated by individual independent set + knapsack instance. Each point is an instance + random seed and points below the diagonal line correspond to better performance than not using bounds.

lem, or plays an important role in the problem. A natural next question is whether this method works well for arbitrary MIP instances. We therefore evaluate our approach on the MIPLIB 2017 benchmark set [32]. The details of this experiment are described in Section B of the appendix. We only observe

	Speed-up (%)	Node reduction (%)
All techniques	66.94	66.10
No Lagrangian relaxation	65.95	65.86
No propagation	60.58	63.50
No Lagrangian relaxation or propagation	46.59	55.69
No primal pruning	60.32	66.03
No primal heuristic	54.33	63.88
No primal pruning or heuristic	54.33	64.01
Without the above techniques	42.39	55.03

Table 3: Effect of removing from consideration generic constraints or primal bounds for the independent set + knapsack instances, averaged across all instances using shifted geometric mean (with shifts of 10 and 100 respectively).

a significant reduction in search tree size for 2 out of the 109 MIPLIB 2017 instances we examined, `mine-166-5` and `mine-90-10`, but even in those cases the overall solving time increased.

There are several reasons that may explain the lack of performance on this benchmark set, and on arbitrary MIP instances in general. The first reason is related to the decision diagram structure—the conflict graph in our case. When the problem instance does not contain a conflict graph, no bound will be generated. Otherwise, when a conflict graph is present, it may generate a weak bound, which may happen in the following cases:

1. The conflict graph captures only a small number of constraints.
2. The convex hull of solutions feasible to the conflict graph is already close to being an integral polyhedron. For instance, a conflict graph may be composed of independent cliques, while other constraints tie them together.
3. The decision diagram relaxation is too weak (e.g., because the conflict graph is sparse).

The second reason is related to the MIP search tree. For example, the MIP solver may not generate sufficiently many search tree nodes of small enough size for the bound generator to be called. Alternatively, the search tree nodes that are pruned by the decision diagram bounds may simply be too small (i.e., represent a small subtree) in order to be effective.

The issues above may potentially be mitigated by considering other substructures within the decision diagram, or by performing a better selection of nodes at which to generate bounds. Despite these negative results on MIPLIB 2017, we emphasize that this method is valuable for instances in which the conflict graph or substructure being leveraged plays an important role in the problem (which may be detected a priori).

8 Conclusion

Relaxed decision diagrams provide good approximations to certain classes of discrete optimization problems, and in this paper we investigate an approach to replicate this power for more general integer programming solvers. We explore approaches to answering two underlying questions in this work: how to construct effective relaxed decision diagrams from integer programming models, and how to use them in order to improve the solving process of a MIP solver.

In this paper, we construct relaxed decision diagrams for a specific substructure of the problem, allowing for DP formulations, merging rules, and variable ordering heuristics that take advantage of that structure. We apply Lagrangian relaxation and constraint propagation to take into account any other constraints. As one possible substructure, we propose the use of the conflict graph, for which we introduce efficient and complete equivalence tests within the construction of a decision diagram.

Once we have a procedure to construct a relaxed decision diagram that heuristically approximates well the feasible set of an integer programming model, the next step is to use it to aid the solving process. We investigate the simple yet effective approach of using dual and primal bounds from these decision diagrams to improve the pruning of the branch-and-bound tree. We find that the bounds are effective both in a case where the entire problem can be modeled as a decision diagram and in a case where it represents a substructure. The decision diagram bounds are able to substantially reduce the tree size in the independent set and independent-set-like instances we tested, leading to a significant improvement in total solving time (roughly 1.6x faster on average in the latter set of instances).

Although we limit our computational experiments to instances in which the conflict graph plays an important role, we provide computational evidence that modeling substructures of a more general problem with decision diagrams can be effective. Handling substructures is particularly important in the case of decision diagrams because, at their current methodological state, they tend to capture well specific structures, but complex combinations of structures are less explored. As decision diagram methodology for optimization evolves, we will be able to better handle different classes of constraints, and our framework allows us to expand the range of problems that decision diagrams can aid in solving.

References

[1] T. Achterberg. “Conflict analysis in mixed integer programming”. In: *Discrete Optimization* 4.1 (2007), pp. 4–20.

[2] T. Achterberg. “Constraint integer programming”. PhD thesis. Technische Universität Berlin, 2009.

[3] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. "Presolve reductions in mixed integer programming". In: *ZIB Report* (2016), pp. 16–44.

[4] S. B. Akers. "Binary decision diagrams". In: *IEEE Transactions on Computers* 100.6 (1978), pp. 509–516.

[5] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. "A constraint store based on multivalued decision diagrams". In: *Principles and Practice of Constraint Programming–CP 2007*. Springer, 2007, pp. 118–132.

[6] B. Aspvall, M. F. Plass, and R. E. Tarjan. "A linear-time algorithm for testing the truth of certain quantified boolean formulas". In: *Information Processing Letters* 8.3 (1979), pp. 121–123.

[7] A. Atamtürk, G. L. Nemhauser, and M. W. Savelsbergh. "Conflict graphs in solving integer programming problems". In: *European Journal of Operational Research* 121.1 (2000), pp. 40–55.

[8] B. Becker, M. Behle, F. Eisenbrand, and R. Wimmer. "BDDs in a branch and cut framework". In: *Experimental and Efficient Algorithms*. Springer, 2005, pp. 452–463.

[9] M. Behle. "Binary decision diagrams and integer programming". PhD thesis. Saarbrücken, Germany: Max Planck Institute for Computer Science, 2007.

[10] F. Benhamou, D. A. McAllester, and P. Van Hentenryck. "CLP(Intervals) Revisited". In: *Proceedings of ILPS*. 1994, pp. 124–138.

[11] D. Bergman and A. A. Cire. "Decomposition based on decision diagrams". In: *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer. 2016, pp. 45–54.

[12] D. Bergman and A. A. Cire. "Discrete nonlinear optimization by state-space decompositions". In: *Management Science* 64.10 (2017), pp. 4700–4720.

[13] D. Bergman, A. A. Cire, and W.-J. van Hoeve. "Improved constraint propagation via lagrangian decomposition". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2015, pp. 30–38.

[14] D. Bergman, A. A. Cire, and W.-J. van Hoeve. "Lagrangian bounds from decision diagrams". In: *Constraints* 20.3 (2015), pp. 346–361.

[15] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. Hooker. *Decision diagrams for optimization*. Springer, 2016.

[16] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. "Discrete optimization with decision diagrams". In: *INFORMS Journal on Computing* 28.1 (2016), pp. 47–66.

[17] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. "Optimization bounds from binary decision diagrams". In: *INFORMS Journal on Computing* 26.2 (2013), pp. 253–268.

[18] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. "Variable ordering for the application of BDDs to the maximum independent set

problem". In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2012, pp. 34–49.

[19] D. Bergman, A. A. Cire, W.-J. van Hoeve, and T. Yunes. "BDD-based heuristics for binary optimization". In: *Journal of Heuristics* 20.2 (2014), pp. 211–234.

[20] D. Bergman, W.-J. van Hoeve, and J. N. Hooker. "Manipulating MDD relaxations for combinatorial optimization". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, 2011, pp. 20–35.

[21] R. E. Bryant. "Graph-based algorithms for boolean function manipulation". In: *IEEE Transactions on Computers* 100.8 (1986), pp. 677–691.

[22] A. A. Cire and W.-J. van Hoeve. "Multivalued decision diagrams for sequencing problems". In: *Operations Research* 61.6 (2013), pp. 1411–1428.

[23] D. Davarnia and W.-J. van Hoeve. "Outer approximation for integer nonlinear programs via decision diagrams". Submitted.

[24] E. Davis. "Constraint propagation with interval labels". In: *Artificial Intelligence* 32.3 (1987), pp. 281–331.

[25] *DIMACS maximum clique benchmark set.* URL: `http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark`.

[26] M. L. Fisher. "An applications oriented guide to Lagrangian relaxation". In: *Interfaces* 15.2 (1985), pp. 10–21.

[27] S. Hoda, W.-J. Van Hoeve, and J. N. Hooker. "A systematic approach to MDD-based constraint programming". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2010, pp. 266–280.

[28] J. Kinable, A. A. Cire, and W.-J. van Hoeve. "Hybrid optimization methods for time-dependent sequencing problems". In: *European Journal of Operational Research* 259.3 (2017), pp. 887–897.

[29] Y.-T. Lai, M. Pedram, and S. B. Vrudhula. "EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13.8 (1994), pp. 959–975.

[30] C.-Y. Lee. "Representation of switching circuits by binary-decision programs". In: *Bell System Technical Journal* 38.4 (1959), pp. 985–999.

[31] L. Lozano and J. C. Smith. "A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs". In: *Mathematical Programming* (2018), pp. 1–24.

[32] *MIPLIB 2017*. http://miplib.zib.de. 2018.

[33] R. J. O'Neil and K. Hoffman. "Decision diagrams for solving traveling salesman problems with pickup and delivery in real time". In: *Operations Research Letters* 47.3 (2019), pp. 197–201.

[34] M. W. Savelsbergh. "Preprocessing and probing techniques for mixed integer programming problems". In: *ORSA Journal on Computing* 6.4 (1994), pp. 445–454.

[35] R. Tarjan. “Depth-first search and linear graph algorithms”. In: *SIAM Journal on Computing* 1.2 (1972), pp. 146–160.

[36] C. Tjandraatmadja and W.-J. van Hoeve. “Target cuts from relaxed decision diagrams”. In: *INFORMS Journal on Computing* (2018). To appear.

[37] D. J. Watts and S. H. Strogatz. “Collective dynamics of ‘small-world’ networks”. In: *Nature* 393.6684 (1998), p. 440.

[38] I. Wegener. *Branching programs and binary decision diagrams: theory and applications*. Vol. 4. SIAM, 2000.

A Additional computational observations

In this section, we report three additional experimental observations on the runs performed in Section 7:

1. Percentage of time spent generating bounds and compiling decision diagrams (the former is the latter plus any Lagrangian relaxation or primal bound computation);
2. Number of improving feasible solutions found by the primal heuristic.
3. Number of times that the bound generator was called (i.e. it met the criterion on subproblem size), that a bound resulted in an improvement over the LP bound at a node, and that the improvement resulted in the node being pruned by bound;

Tables 4 through 9 in this section report the values described above. All averages of the values above are done with arithmetic mean across the instances and random seeds examined in Section 7. Note that the percentage of time spent on each portion of the method are calculated via ratios of times summed up across all seeds and instances (or equivalently, via ratios of arithmetic means).

B Experiments on MIPLIB 2017

We performed the following simple experiment on MIPLIB 2017 instances [32]. We took the 109 instances with the tags ‘binary‘ and ‘benchmark_suitable‘ from MIPLIB 2017. For these instances, we ran SCIP with bounds from relaxed decision diagrams, using a width of 100 and generating them at subproblems where number of variables is at most 1000. We chose a threshold that is not problem-dependent to avoid building large decision diagrams in large problems.

We ran a first pass with the intention to filter out instances where our approach is very unlikely to help. For this pass, we used a time limit of 30 minutes and one random seed. Out of the 109 instances, there were only 38 of them where the bound generator was called, or in other words, the solver did not observe a subproblem of size at most 1000 within the time limit of 30 minutes. Out of these 38 instances, in only 9 instances the bound generator found a bound better than the LP bound at least 20 times. These were the following 9 instances: `bnatt400`, `eil33-2`, `eilC76-2`, `mine-166-5`, `mine-90-10`, `neos18`, `ponderthis0517-inf`, `reblock115`, `reblock166`.

In the second pass, we reran the same experiment for these 9 instances, but with five random seeds and a higher time limit of 3 hours. We analyze the 4 instances of this subset that did not hit the time limit for the runs with bounds from decision diagrams. We report the solving times and number of nodes in Table 10, and number of bound improvements (over LP) and pruning in Table 11.

In all cases, the overhead of generating bounds increases the solving time, sometimes substantially. However, for the instances `mine-166-5` and `mine-90-10`, we do obtain relevant reductions in the number of nodes by 25.97% and 23.77% respectively. Moreover, we observe that in all four instances, a significant portion of bounds generated improve over the corresponding LP bounds.

Density (%)	Percentage of time on bound generation (%)	Percentage of time on DD compilation (%)	Number of primal improvements
10	68.61	65.07	2.26
20	60.90	58.72	3.15
30	39.96	38.33	2.31
40	15.61	14.99	1.75
50	2.72	2.69	1.31
60	0.88	0.87	1.41
70	0.47	0.46	1.47
80	0.20	0.20	0.30
90	0.09	0.09	0.31

Table 4: Percentage of time used for bound generation and number of times an improving primal feasible solution was found for independent set instances of size 150 from Section 7.1.

Density (%)	Number of calls to bound generator	Number of improvements	Number of prunings
10	5300.21	5041.30	5041.30
20	5912.15	3051.69	3051.69
30	915.35	423.02	423.02
40	199.87	75.02	75.02
50	54.77	2.95	2.95
60	50.65	1.32	1.32
70	46.36	1.52	1.52
80	18.42	0.29	0.29
90	1.44	0.31	0.31

Table 5: Number of times an improving bound was found or resulted in pruning for independent set instances of size 150 from Section 7.1.

Density (%)	Percentage of time on bound generation (%)	Percentage of time on DD compilation (%)	Number of primal improvements
40	35.67	33.98	3.04
50	20.26	19.11	2.50
60	5.20	4.94	1.55
70	0.72	0.71	1.74
80	0.35	0.35	0.37
90	0.07	0.07	0.62

Table 6: Percentage of time used for bound generation and number of times an improving primal feasible solution was found for independent set instances of size 300 from Section 7.1.

Density (%)	Number of calls to bound generator	Number of improvements	Number of prunings
40	11296.06	6692.04	5686.14
50	1808.46	1072.50	953.09
60	245.09	184.12	172.39
70	107.77	104.52	102.65
80	98.34	98.34	97.96
90	7.35	7.34	6.72

Table 7: Number of times an improving bound was found or resulted in pruning for independent set instances of size 300 from Section 7.1.

Graph size	Percentage of time on bound generation (%)	Percentage of time on DD compilation (%)	Number of primal improvements
300	2.87	2.86	0.97
350	5.23	5.19	1.09
400	8.20	8.20	1.46
450	10.17	10.16	2.19

Table 8: Percentage of time used for bound generation and number of times an improving primal feasible solution was found for independent set + knapsack instances from Section 7.2.

Graph size	Number of calls to bound generator	Number of improvements	Number of prunings
300	192.86	190.51	188.87
350	567.40	564.95	563.30
400	1940.51	1939.24	1937.21
450	5639.30	5636.89	5633.71

Table 9: Number of times an improving bound was found or resulted in pruning for independent set + knapsack instances from Section 7.2.

	Solving time (s)		Number of nodes	
Instance	MIP	MIP + DD	MIP	MIP + DD
`eil33-2`	213.89	235.33	328.55	334.14
`mine-166-5`	161.46	664.70	2630.97	1947.59
`mine-90-10`	1119.07	7197.87	30864.59	23528.99
`neos18`	122.96	5084.48	2100.63	1972.28

Table 10: Solving time in seconds and number of nodes for a subset of MIPLIB 2017 instances.

Density (%)	Number of calls to bound generator	Number of improvements	Number of prunings
`eil33-2`	213.8	163.6	156.8
`mine-166-5`	1763.4	337.2	188.0
`mine-90-10`	20593.2	3244.0	2162.0
`neos18`	1655.8	152.4	112.4

Table 11: Number of times an improving bound was found or resulted in pruning for a subset of MIPLIB 2017 instances.