Cost-Bounded Binary Decision Diagrams for 0-1 Programming

Tarik Hadžić[1] and J. N. Hooker[2]

[1] IT University of Copenhagen
tarik@itu.dk
[2] Carnegie Mellon University
john@hooker.tepper.cmu.edu

Abstract. In recent work binary decision diagrams (BDDs) were introduced as a technique for postoptimality analysis for integer programming. In this paper we show that much smaller BDDs can be used for the same analysis by employing cost bounding techniques in their construction.

Binary decision diagrams (BDDs) have seen widespread application in logic circuit design and product configuration. They also have potential application to optimization, particularly to postoptimality analysis. A BDD can represent, often in compact form, the entire feasible set of an optimization problem. Optimal solutions correspond to shortest paths in the BDD. Due to the efficiency of this representation, one can rapidly extract a good deal of information about a problem and its solutions by querying the BDD.

This opens the door to fast, in-depth postoptimality analysis without having to re-solve the problem repeatedly—provided the BDD is of manageable size. For instance, one can identify all optimal or near-optimal solutions by finding all shortest or near-shortest paths in the BDD. One can quickly determine how much freedom there is to alter the solution without much increase in cost. This is particularly important in practice, since managers often require some flexibility in how they implement a solution. One can deduce, in real time, the consequences of fixing a variable to a particular value. One can conduct several types of sensitivity analysis to measure the effect of changing the problem data.

We present in [1] several techniques for postoptimality analysis using BDDs. We address here a key computational issue: how large does the BDD grow as the problem size increases, and how can one minimize this growth? In particular we examine the strategy of generating a BDD that represents only near-optimal solutions, since these are generally the solutions of greatest interest in practice. In principle, a BDD that exactly represents the set of near-optimal solutions need be no smaller than one that represents all solutions, and it can in fact be exponentially larger. At least in the problem domain we investigate, however, the BDD representing near-optimal solutions is significantly smaller. We also identify a family of *sound* BDDs that are even smaller but support valid postoptimality analysis—even though they do not exactly represent the set of

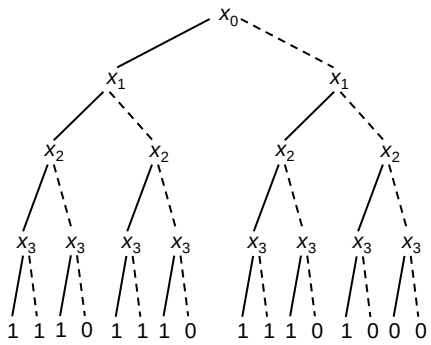

Fig. 1. Branching tree for $2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$

near-optimal solutions. This allows for postoptimality analysis of much larger problem instances by using sound BDDs.

One advantage of BDD-based analysis is that it presupposes no structure in the problem, aside from separability of the objective function and finiteness of domains. The methods are the same regardless of whether the constraint and objective functions are linear, nonlinear, convex, or nonconvex. However, for purposes of experimentation we focus on 0-1 linear programming problems. The methods described here are readily extended to nonlinear constraints. They can also be extended to general integer variables by writing each variable as a vector of 0-1 variables.

1 Binary Decision Diagrams

A BDD is a directed graph that represents a boolean function. A given boolean function corresponds to a unique *reduced BDD* when the variable ordering is fixed. The same is true of a constraint set in 0-1 variables, since it can be viewed as a boolean function that is true when the constraints are satisfied and false otherwise.

The reduced BDD is essentially a compact representation of the branching tree for the constraint set. The leaf nodes of the tree are labelled by 1 or 0 to indicate that the constraint set is satisfied or violated. For example, the tree of Fig. 1 represents the 0-1 linear inequality

$$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7 \ . \tag{1}$$

The solid branches (*high edges*) correspond to setting $x_j = 1$ and the dashed branches (*low edges*) to setting $x_j = 0$.

The tree can be transformed to a reduced BDD by repeated application of two operations: (a) if both branches from a node lead to the same subtree, delete the node; (b) if two subtrees are identical, superimpose them. The reduced BDD for (1) appears in Fig. 2(a).

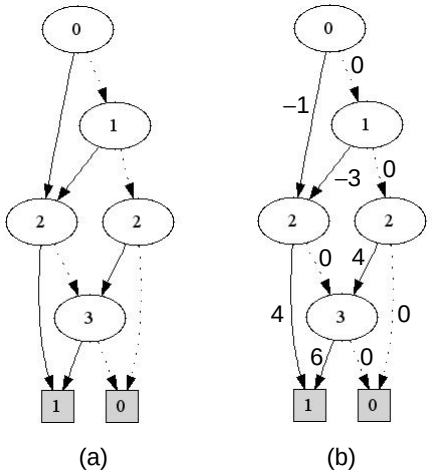

Fig. 2. (a) Reduced BDD for $2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$ using the variable ordering x_0, x_1, x_2, x_3. (b) Same BDD with edge lengths corresponding to the objective function $2x_0 - 3x_1 + 4x_2 + 6x_3$

Each path from the root to 1 in a BDD *represents* one or more solutions, namely all solutions in which x_j is set to 0 when the path contains a low edge from a node labelled x_j, and is set to 1 when the path contains a high edge from such a node. A BDD B represents the set $Sol(B)$ of all solutions that are represented by a path from the root to 1.

A reduced BDD can in principle be built by constructing the search tree and using intelligent caching to eliminate nodes and superimpose isomorphic subtrees. It is more efficient in practice, however, to combine the BDDs for elementary components of the boolean function. For example, if there are several constraints, one can build a BDD for each constraint and then conjoin the BDDs. Two BDDs can be conjoined in time that is roughly quadratic in the number of BDD nodes. Thus if the individual constraints have compact BDDs, this structure is exploited by constructing search trees for the individual constraints and conjoining the resulting BDDs, rather than constructing a search tree for the entire constraint set. Algorithms for building reduced BDDs in this fashion are presented in [2, 1].

The BDD for a linear 0-1 inequality can be surprisingly compact. For instance, the 0-1 inequality

$$\begin{aligned} &300x_0 + 300x_1 + 285x_2 + 285x_3 + 265x_4 + 265x_5 + 230x_6 + \\ &\quad 23x_7 + 190x_8 + 200x_9 + 400x_{10} + 200x_{11} + 400x_{12} + \\ &\quad 200x_{13} + 400x_{14} + 200x_{15} + 400x_{16} + 200x_{17} + 400x_{18} \geq 2701 \end{aligned} \tag{2}$$

has a complex feasible set that contains 117,520 minimally feasible solutions (each of which becomes infeasible if any variable is flipped from 1 to 0), as

reported in [3]. (Equivalently, if the right-hand side is ≤ 2700, the inequality has 117,520 minimal covers.) The BDD for (2) contains only 152 nodes.

A separable objective function $\sum_j c_j(x_j)$ can be minimized subject to a constraint set by finding a shortest path from the root to 1 in the corresponding BDD. If node u and u' have labels x_k and x_ℓ, respectively, then a high edge from u to u' has length $c^1[u,u'] = c_k(1) + c^*_{k+1,\ell-1}$, where $c^*_{k+1,\ell-1}$ is the cost of setting every skipped variable $x_{k+1},\ldots,x_{l-1}$ to a value that gives the lowest cost. More precisely:

$$c^v[u,u'] = c_k(v) + c^*_{k+1,\ell-1}$$

and

$$c^*_{pq} = \sum_{j=p}^{q} \min\{c_j(1), c_j(0)\}$$

A low edge from u to u' has length $c^0[u,u']$. For example, if we minimize

$$2x_0 - 3x_1 + 4x_2 + 6x_3 \tag{3}$$

subject to (1), the associated BDD has the edge lengths shown in Fig. 2(b). Note that the length of the high edge from the root node is $c_0(1) + c^*_{11} = 2 - 3 = -1$. The shortest path from the root node to 1 has length 1 and passes through the x_1 node and the x_2 node on the left. Its three edges indicate that $(x_0, x_1, x_2) = (0, 1, 1)$. This corresponds to optimal solution $(x_0, x_1, x_2, x_3) = (0, 1, 1, 0)$, where x_3 is set to zero to minimize the x_3 term in the objective function.

2 Previous Work

BDDs have been studied for decades [4, 5]. Bryant [6] showed how to reduce a BDD to a unique canonical form, for a given variable ordering. Readable introductions to BDDs include [2, 7].

There has been very little research into the application of BDDs to optimization. Becker at al. [8] used BDDs to identify separating cuts for 0-1 linear programming problems in a branch-and-cut context. They generated BDDs for a subset of constraints and obtained a cut $ux \geq u_0$ that is violated by the solution $\bar{x}$ of the linear relaxation of the problem. The cut is obtained by using subgradient optimization to find an assignment of costs u_i to edges of the BDD for which $ux < u_0$, where u_0 is the length of a shortest path to 1 in the BDD.

In [1] we show how BDDs can be used for various types of postoptimality analysis. One type is cost-based domain analysis, which computes the projection of the set of near-optimal solutions onto any given variable. Near-optimal solutions are those whose objective function value is within Δ of the optimal value, where Δ is specified by the user. This type of analysis shows the extent to which the values of variables can be changed without increasing cost more than Δ. We also show how to perform conditional domain analysis, which computes the projections after restricting the domain of one or more variables. This shows the consequences of making certain decisions on possible values for the remaining

variables. We illustrate these techniques on capital budgeting, network reliability, and investment problems, the last two of which are nonlinear and nonconvex, and all of which involve general integer variables.

3 Projection and Postoptimality Analysis

Consider a 0-1 programming problem with a separable objective function:

$$\begin{aligned} &\min \sum_{j=1}^{n} c_j(x_j) \\ &g_i(x) \geq b_i, \quad i = 1, \ldots, m \\ &x_j \in \{0,1\}, \quad j = 1, \ldots, n \end{aligned} \tag{4}$$

We refer to any 0-1 n-tuple as a *solution* of (4), any solution that satisfies the constraints as a *feasible solution*. If Sol is the set of feasible solutions, let Sol_j be the projection of Sol onto variable x_j. Then Sol_j can be easily deduced from the reduced BDD B of (4). Sol_j contains the value 0 if at least one path from the root to 1 in B contains a low edge from a node labelled x_j, and it contains 1 if at least one path contains a high edge from a node labelled x_j.

If c^* is the optimal value of (4), then for a given tolerance Δ the set of near-optimal feasible solutions of (4) is

$$Sol_\Delta = \left\{ x \in Sol \;\middle|\; \sum c_j(x_j) \leq c^* + \Delta \right\}$$

In general, *cost-based domain analysis* derives the projection $Sol_{\Delta j}$ of Sol_Δ onto any x_j for any Δ between 0 and some maximum tolerance $\Delta_{\max}$. It may also incorporate conditional domain analysis subject to a partial assignment $x_J = v_J$ specified by the user, where $J \subset \{1, \ldots, n\}$. This projection is

$$Sol_{\Delta j}(x_J = v_J) = \{x_j \mid x \in Sol_\Delta, x_J = v_J\}$$

Algorithms for implementing cost-based domain analysis are presented in [1], and they are based on compiling the solution set of (4) into a BDD B. $Sol_{\Delta j}$ can be efficiently computed by examining paths of length at most $c^* + \Delta$ from the root to 1 in B. $Sol_{\Delta j}$ contains 0 if at least one such path contains a low edge from a node labelled x_j, and it contains 1 if at least one such path contains a high edge from a node labelled x_j [9]. Similar analysis for $Sol_{\Delta j}(x_J = v_J)$ is performed on a restricted BDD that can be efficiently constructed from B.

It is obvious that domain-analysis is correct if B represents exactly the solution set Sol. However, the main computational issue is obtaining a BDD B of manageable size. It is therefore useful to find smaller BDDs that do not necessarily represent Sol but still yield the same outputs $Sol_{\Delta j}$. We will say that any BDD B' over which the algorithms from [1] compute the required $Sol_{\Delta j}$ *yields correct cost-based domain analysis.*

4 Cost-Bounded BDDs

To obtain correct cost-based domain analysis from a smaller BDD, a straightforward strategy is to use a BDD that represents only near-optimal solutions. A BDD that represents all solutions is not necessary, since $Sol_{\Delta j}$ contains projections of near-optimal solutions only. Thus if B represents the solution set of (4), we denote by $B_{cx\leq b}$ the BDD representing the set of near-optimal solutions, where $b = c^* + \Delta_{max}$. $B_{cx\leq b}$ can be built by adding $cx \leq b$ to the constraint set of (4) and constructing a BDD in the usual fashion. We will refer to this exact representation of near-optimal solution set as an *exact BDD*.

Although $Sol(B_{cx\leq b})$ is smaller than $Sol(B)$, $B_{cx\leq b}$ is not necessarily smaller than B. In fact, it can be exponentially larger. For example, consider a 0-1 programming model (4) over $n = p^2$ variables $\{x_{kl} \mid k, l = 1, \ldots, p\}$. Let there be no constraints, i.e., the solution set contains all 0-1 tuples, so that resulting BDD B is a single 1 node (a tautology). Let the objective function of (4) be

$$\sum_{k=1}^{p}\sum_{\ell=1}^{p} c_{k\ell}x_{k\ell}$$

where $c_{k\ell} = 2^{k-1} + 2^{\ell+p-1}$, so that $c^* = 0$. If we let

$$\bar{b} = \tfrac{1}{2}\sum_{k=1}^{p}\sum_{\ell=1}^{p} c_{k\ell} = \tfrac{1}{2}p(2^{2p} - 1)$$

then Theorem 6 from [10] states that a BDD B_1 representing function $cx \geq \bar{b}$ has the width of at least $\Omega(2^{\sqrt{n}/2})$ and is therefore exponentially large. A BDD B_2 representing $cx \leq \bar{b} - 1$ is a negation of B_1, obtained by just swapping terminal nodes 0 and 1, and has the same width $\Omega(2^{\sqrt{n}/2})$. Therefore, if we take $b = \bar{b} - 1$, $B_{cx\leq b}$ is exponentially larger than B.

On the other hand, exact BDD can also be exponentially smaller than B. For example, if the objective function is $\sum_{k\ell} x_{k\ell}$ and the constraint set consists of $\sum_{k\ell} c_{k\ell}x_{k\ell} \leq \bar{b} - 1$, then $c^* = 0$ and B has the width $\Omega(2^{\sqrt{n}/2})$. However, $Sol(B_{cx\leq b})$ contains one solution when $b = c^* = 0$, namely $x = 0$, and $B_{cx\leq b}$ therefore has a linear number of nodes.

5 Sound BDDs

As part of a strategy for overcoming the possible size explosion of an exact BDD, we suggest a family of *sound* BDDs to be used for cost-based domain analysis. A sound BDD for a given Δ_{max} is any BDD B' for which

$$Sol(B') \cap \{x \in \{0,1\}^n \mid cx \leq b\} = Sol(B_{cx\leq b}) \tag{5}$$

where again $b = c^* + \Delta_{max}$. Clearly,

Lemma 1. *Any sound BDD yields correct cost-based domain analysis.*

This is because if B' is sound, the elements of B' with cost at most b are precisely the elements of $Sol(B)$ with cost at most b. Thus when B' is used to compute $Sol_{\Delta j}$, the result is the same as when using $B_{cx \leq b}$ or B. Note that $Sol(B')$ need not be a subset of $Sol(B)$. We can add or remove from $Sol(B')$ any element with cost greater than b without violating soundness.

A smallest sound BDD is no larger than either B or $B_{cx \leq b}$ (as both are sound themselves), and it may be significantly smaller than both.

We are unaware of a polynomial-time exact algorithm for computing a smallest sound BDD, but we offer a heuristic method that uses two polynomial-time operations, *pruning* and *contraction*, each of which reduces the size of a BDD while preserving soundness.

6 Pruning

Pruning a BDD removes edges that belong only to paths that are longer than the cost bound b. Pruning therefore reduces the size of the BDD without removing any solutions with cost less than or equal to b.

Define a path from the root to 1 to be *admissible* with respect to b if it represents at least one solution x with $cx \leq b$. An edge in the BDD is admissible if it lies in at least one admissible path. *Pruning* is the operation of removing an inadmissible edge. A BDD is *pruned* if it contains no inadmissible edges.

Pruning clearly preserves soundness, but some pruned and sound BDDs for the same constraint set may be smaller than others. Consider BDDs in Fig. 3 defined over two variables x_1 and x_2 and the objective function $x_1 + x_2$. Then if $b = 1$, both BDDs in Fig. 3 are pruned and sound.

Fig. 4 displays an algorithm that generates a pruned BDD, given a starting BDD B and a cost bound $cx \leq b$ as input. In the algorithm, $L[j]$ is the set of nodes of B with label x_j. Pruning starts with the last layer of nodes $L[n-1]$ and proceeds to first layer $L[0]$. Each round of pruning creates a new B, which is stored as B_{old} before the next round starts. For a given node u, $l(u)$ and $h(u)$ are its low and high children, respectively, in B_{old}. If node u has label x_j, the algorithm checks whether the low edge $(u, l(u))$ is too expensive by checking whether the shortest path from the root to 1 through that edge is longer than b. If the edge $(u, l(u))$ is too expensive, it is deleted by redirecting it to a terminal node 0, i.e., by replacing it with $(u, 0)$. A similar test is performed for the high edge.

The algorithm checks whether edge $(u, l(u))$ is too expensive by checking whether $U[u] + c^0[u, l(u)] + D[l(u)] > b$, where $c^0[u, l(u)]$ is the length of the edge, $U(u)$ is the length of the shortest path from u up to the root in B_{old}, and $D(l(u))$ is the length of the shortest path from $l(u)$ down to 1 in B_{old}. By convention, $D(0) = \infty$. Replacement of $(u, l(u))$ with $(u, 0)$ is implemented by calls to a standard BDD node creation function that ensures that the resulting BDD is reduced [1].

When $\Delta_{\max} = 0$, pruning retains only edges that belong to a shortest path. If all edges in a BDD belong to a shortest path from the root to 1, then all paths

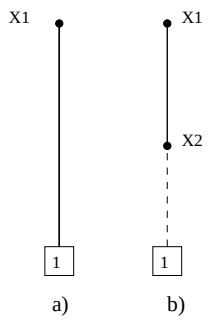

Fig. 3. Two pruned and sound BDDs over variables x_1, x_2. If $b = 1$, each edge in (a) and (b) is part of an admissible path representing the solution $(x_1, x_2) = (1, 0)$.

Function **prune**(B, c, b)
  $B_{old} \leftarrow 0$
  While $B \neq B_{old}$
    $B_{old} \leftarrow B$, update $D[\cdot], U[\cdot]$
    For $j = n - 1$ to 0:
      For $u \in L[j]$:
        If $U[u] + c^0[u, l(u)] + D[l(u)] > b$ then
          replace $(u, l(u), h(u))$ with $(u, 0, h(u))$ in B
        Else if $U[u] + c^1[u, h(u)] + D[h(u)] > b$ then
          replace $(u, l(u), h(u))$ with $(u, l(u), 0)$ in B
        $D[u] \leftarrow min\{c^0[u, l(u)] + D[l(u)], c^1[u, h(u)] + D[h(u)]\}$
  Return B.

Fig. 4. Algorithm for pruning a BDD B with respect to $cx \leq b$.

from the root to 1 are shortest paths, due to the Lemma 2. As a consequence, the number of all paths in a pruned BDD is bounded by the number of optimal solutions, as every path represents at least one optimal solution.

Lemma 2. *If every edge in a directed acyclic graph G belongs to a shortest source-terminus (s-t) path, then every s-t path in G is shortest.*

Proof. Suppose to the contrary there is an s-t path P in G that is not shortest. Let P' be the shortest subpath of P that is part of no shortest s-t path. Then P' contains at least two edges, which means that P' can be broken into two subpaths A and B, and we write $P' = A + B$ (Fig. 5). Since P' is minimal, A is part of a shortest s-t path $A_s + A + A_t$, and B is part of a shortest s-t path $B_s + B + B_t$. But

$$B_s < A_s + A \tag{6}$$

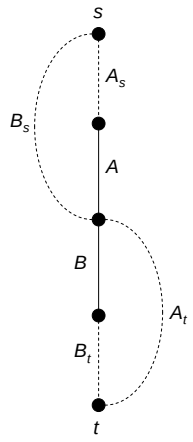

Fig. 5. Illustration of Lemma 2.

(where $<$ means "is shorter than"), since otherwise $A + B$ is part of a shortest s-t path $A_s + A + B + B_t$. But (6) implies

$$B_s + A_t < A_s + A + A_t$$

which contradicts the fact that $A_s + A + A_t$ is a shortest path.

Even more, when $\Delta_{\max} = 0$ we can efficiently construct exact BDD $B_{cx \leq b}$ from the pruned BDD B_Δ. Whenever an edge (u_1, u_2) skips a variable x_i, and if a cost of $c_i(0)$ is cheaper than $c_i(1)$, we forbid assignment $x_i = 1$ by inserting a node u_i that breaks the edge (u_1, u_2) into (u_1, u_i) and (u_i, u_2) in such a way that the low child of u is u_2, $l(u_i) = u_2$, while the high child is 0, $h(u_i) = 0$. This operation increases the number of nodes by 1. Hence, it suffices for every skipping edge and every skipped variable covered by that edge to insert a node to get a BDD B'_Δ that represents exactly the set of optimal solutions. The exact BDD $B_{cx \leq b}$ is obtained by reducing B'_Δ.

It follows that exact BDD cannot be exponentially bigger than pruned BDD when $\Delta_{\max} = 0$. A simple overestimation gives us a bound on the number of nodes: $|B_{cx \leq b}| \leq |B_\Delta| + |E| \cdot n$, where E is the set of edges in B_Δ. Namely, for each edge in E we can insert at most n nodes.

When $\Delta_{\max} > 0$, even when all inadmissible edges are removed, some paths of length less than or equal to b may remain. This is because even if every edge of a graph belongs to an s-t path of length at most b, the graph may yet contain an s-t path longer than b. Consider for example the graph of Fig. 6. Every edge belongs to an s-t path with length at most 4, but there is nonetheless an s-t path of length 6.

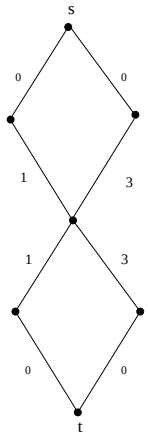

Fig. 6. Example for pruning when $\Delta_{max} > 0$. Even though every edge belongs to a path of length at most 4, there are paths of length 6.

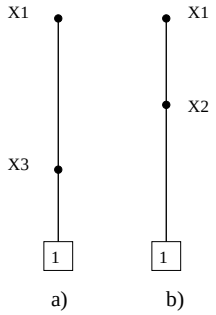

Fig. 7. Two contracted and sound BDDs over variables x_1, x_2, x_3 and objective function $x_1 + x_2 + x_3$. If $b = 1$, the contraction of a node introduces a solution $(x_1, x_2, x_3) = (1, 0, 0)$ with cost 1.

7 Contraction

A node u in a BDD B is *contractible* with respect to parent node u_p, child node u_c, and cost restriction $cx \leq b$ if replacing edges (u_p, u) and (u, u_c) with an edge (u_p, u_c) introduces no solutions with cost less than or equal to b. That is, if B^c is the BDD resulting from the replacement, $cx > b$ for all $x \in Sol(B^c) \setminus Sol(B)$. To *contract* node u is to replace (u_p, u) and (u, u_c) with (u_p, u_c). A BDD is *contracted* if it contains no contractible nodes.

Contraction obviously preserves soundness, as it adds only solutions with cost greater than b. Yet as with pruning, there are more than one sound and contracted BDD for the same constraint set. Consider BDDs in Fig. 7 defined over variables x_1, x_2, and x_3, and objective function $x_1 + x_2 + x_3$. If $b = 1$, both BDDs are sound and contracted.

```
Function contract(B, c, b)
    B_old ← 0
    While B ≠ B_old
        B_old ← B, update D[·], U[·]
        For j = n − 1 to 0
            For all u ∈ L[j]
                If l(h(u)) = 0 then
                    If U[u] + c^1[u, h(u)] + c^0[h(u), h(h(u))] + D[h(h(u)] > b then
                        Replace (u, h(u)), (h(u), h(h(u))) with (u, h(h(u))) in B
                If h(h(u)) = 0 then
                    If U[u] + c^1[u, h(u)] + c^1[h(u), l(h(u))] + D[l(h(u)] > b then
                        Replace (u, h(u)), (h(u), l(h(u))) with (u, l(h(u))) in B
                If l(l(u)) = 0 then
                    If U[u] + c^0[u, l(u)] + c^0[l(u), h(l(u))] + D[h(h(u)] > b then
                        Replace (u, l(u)), (l(u), h(l(u))) with (u, h(l(u))) in B
                If h(l(u)) = 0 then
                    If U[u] + c^0[u, l(u)] + c^1[h(u), l(l(u))] + D[l(l(u)] > b then
                        Replace (u, l(u)), (l(u), l(l(u))) with (u, l(l(u))) in B
    Return B
```

Fig. 8. Algorithm for contracting a BDD B with respect to cost bound $cx \leq b$.

An algorithm that implements contraction is presented in Fig. 8. The algorithm repeatedly contracts nodes u with only one non-zero child, until no contracting is possible.

Lemma 3. *The algorithm of Fig. 8 removes only contractible nodes.*

Proof. Suppose that the algorithm contracts node $h(u)$ by replacing $(u, h(u))$ with $(u, h(h(u))$. It suffices to show that the replacement adds no solutions with cost less than or equal to b. (The argument is similar for the other three cases.) Suppose that u has label x_k, $h(u)$ has label x_ℓ, and $h(h(u))$ has label x_m. Any path through the new edge $(u, h(h(u)))$ represents a solution that is also represented by a path through the original edges $(u, h(u))$ and $(h(u), h(h(u)))$, unless $x_\ell = 0$. So we need only check that any solution with $x_\ell = 0$ represented by a path through $(u, h(h(u)))$ has cost greater than b. But the cost of any such solution is at least

$$\begin{aligned} &U[u] + c_k(1) + c^*_{k+1,\ell-1} + c_\ell(0) + c^*_{\ell+1,m-1} + D[h(h(u))] \\ &= U[u] + c^1[u, h(u)] + c^0[h(u), h(h(u))] + D[h(h(u))] \end{aligned}$$

The algorithm ensures that node $h(u)$ is not contracted unless this quantity is greater than b.

Both pruning and contraction algorithm have worst-case running time that is quadratic in the size of the underlying BDD. Both algorithms explore all the

```
Function compile(b)
    B_Δ ← 1
    for i = 1 to m
        B_i ← BDD(g_i)
        B_Δ ← B_Δ ∧ B_i
        if (|B_Δ| > T or i = m)
            B_Δ ← prune(B_Δ, c, b)
            B_Δ ← contract(B_Δ, c, b)
    return B_Δ
```

Fig. 9. A simple compilation scheme to obtain a sound BDD, for problem (4) and cost bound b, that is pruned and contracted.

BDD nodes in each internal iteration. There is at most linear number of these iterations, since each time at least one edge is removed.

Fig. 9 presents simple algorithm for compiling a pruned and contracted BDD for (4). The variable ordering is fixed to $x_1 < \ldots < x_n$, and BDD(g_i) denotes an atomic compilation step that generates a BDD from the syntactical definition of $g_i(x) \geq b_i$. Pruning and contraction are applied not only to the final BDD, but to intermediate BDDs when their size exceeds a threshold T.

8 Computational Results

We carried out a number of experiments to analyze the effect of cost bounding on the size of BDDs. In particular, we wish to test the benefits of replacing an exact cost-bounded BDD with a sound BDD obtained by pruning and contraction.

We performed experiments over randomly generated 0-1 linear programs with multiple inequalities and a few 0-1 instances from the MIPLIB library.[3] All the experiments were carried out on a Pentium-III 1 GHz machine with 2GB of memory, running Linux. To load instances we used customized version of a BDD-based configuration library CLab [11], running over BDD package BuDDy [12].

For each instance we compared three BDDs: the BDD B representing the original constraint set, the exact bounded BDD $B_{cx\leq b}$, and the sound BDD B_Δ that results from pruning and contracting B. We show results for several values of Δ. The optimal value c^* and the largest feasible value $c_{\max}$ of the objective function are shown for comparison.

The results show that both $B_{cx\leq b}$ and B_Δ are substantially smaller than B for rather large cost tolerances Δ. Also B_Δ is almost always significantly smaller than $B_{cx\leq b}$. For example, in an instance with 30 variables and six constraints, one can explore all solutions within a range of 50 of the optimal value 36 by constructing a sound BDD that is only a tiny fraction of the size the full BDD

[3] Available at http://miplib.zib.de/miplib2003.php

Table 1. Experimental results for 0-1 linear programs. Each instance has n variables and m constraints. The coefficients a_j of 0-1 inequalities $\sum_{j=1}^{n} a_j x_j \geq b$ are drawn uniformly from $[0, r]$, and b is chosen such that $b = \alpha \cdot \sum_{j=1}^{n} a_j$ where α indicates the tightness of the constraints. The optimal value is c^*, and the largest feasible objective function value is $c_{\max}$. The size of the original BDD B is not shown when it is too large to compute.

$n = 20$, $m = 5$
$r = 50$, $\alpha = 0.3$
$c^* = 101$, $c_{\max} = 588$

Δ	$\lvert B_\Delta \rvert$	$\lvert B_{cx \leq b} \rvert$	$\lvert B \rvert$
0	5	20	8566
40	524	742	8566
80	3456	4328	8566
120	7037	11217	8566
200	8563	16285	8566
240	8566	13557	8566

$n = 30$, $m = 6$
$r = 60$, $\alpha = 0.3$
$c^* = 36$, $c_{\max} = 812$

Δ	$\lvert B_\Delta \rvert$	$\lvert B_{cx \leq b} \rvert$	$\lvert B \rvert$
0	10	30	925610
50	2006	3428	925610
150	262364	226683	925610
200	568863	674285	925610
250	808425	1295465	925610
200	905602	1755378	925610

$n = 40$, $m = 8$
$r = 80$, $\alpha = 0.3$
$c^* = 110$, $c_{\max} = 1241$

Δ	$\lvert B_\Delta \rvert$	$\lvert B_{cx \leq b} \rvert$
0	12	40
15	402	1143
35	1160	3003
70	7327	11040
100	223008	404713
140	52123	

$n = 50$, $m = 5$
$r = 100$, $\alpha = 0.1$
$c^* = 83$, $c_{\max} = 2531$

Δ	$\lvert B_\Delta \rvert$	$\lvert B_{cx \leq b} \rvert$	$\lvert B \rvert$
0	12	83	4891332
100	103623	163835	4891332
200	1595641	2383624	4891332

$n = 60$, $m = 10$
$r = 100$, $\alpha = 0.1$
$c^* = 67$, $c_{\max} = 3179$

Δ	$\lvert B_\Delta \rvert$	$\lvert B_{cx \leq b} \rvert$
0	7	60
50	1814	5519
100	78023	111401

(2006 nodes versus 925,610 nodes). In problems with 40 and 60 variables, the full BDD is too large to compute, while sound BDDs are of easily manageable size for a wide range of objective functions values.

Fig. 10 illustrates how $|B_{cx \leq b}|$ and $|B_\Delta|$ compare over the full range of objective function values. Note that $|B_{cx \leq b}|$ is actually larger than $|B|$ for larger values of Δ, even though $B_{cx \leq b}$ represents fewer solutions than B. The important fact for postoptimality analysis, however, is that $|B_{cx \leq b}|$, and especially $|B_\Delta|$, are much smaller than $|B|$ for small Δ.

Table 2 shows the results for a few 0-1 problems in MIPLIB for fixed $\Delta = 0$. A BDD for $\Delta = 0$ is useful for identifying all optimal solutions. We observe significant savings in space for both B_Δ and $B_{cx \leq b}$ in comparison to B. For instance *lseu*, we were not able to generate B, while both B_Δ and $B_{cx \leq b}$ are quite small.

The response times of algorithms for calculating valid domains and postoptimality analysis depend linearly on the size of an underlying BDD. In our experience, response time over a BDD having 10 000 nodes is within tens of milliseconds. A response time of up to one second corresponds to a BDD with about 250 000 nodes. In terms of memory consumption, one BDD node takes 20 bytes of memory, hence 50 000 nodes take 1 MB.

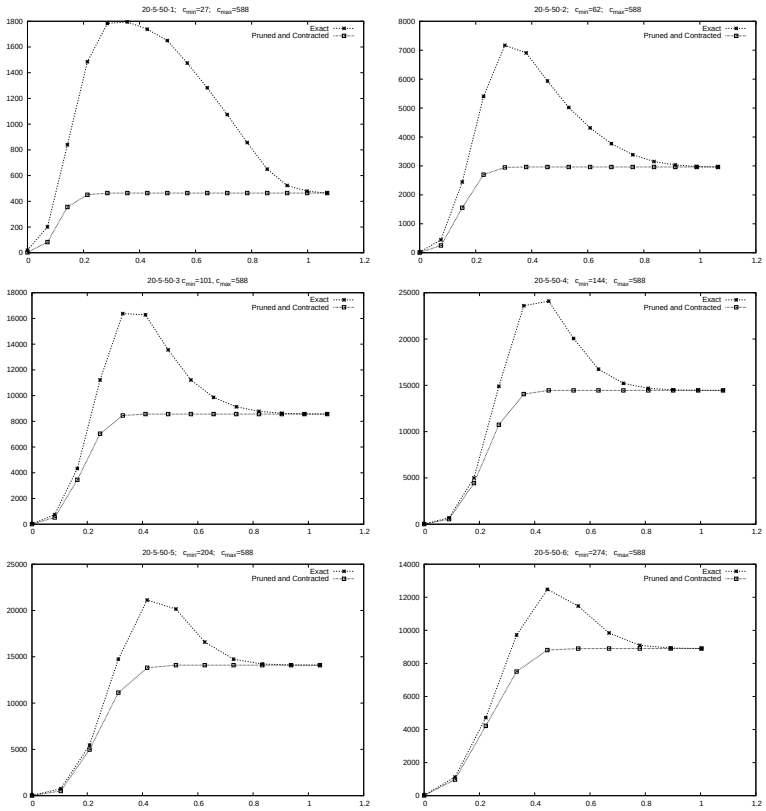

Fig. 10. Illustration of how the size of exact BDDs $B_{cx\leq b}$ (upper curve) and pruned and contracted BDDs B_{Δ} (lower curve) in 0-1 linear instances depends on Δ. Here there are 20 variables, 5 constraints, and $r = 50$. The horizontal axis indicates Δ as a fraction of $c_{\max} - c^*$. The tightness parameter α takes the values $0.1, 0.2, \ldots, 0.6$ in the six plots. The difference between $|B_{cx\leq b}|$ and $|B_{\Delta}|$ is larger when the constraints are more relaxed (α is small).

9 Conclusion

We conclude that cost-bounded BDDs can yield significant computational advantages for cost-based domain analysis of 0-1 linear programming problems of moderate size. Sound BDDs obtained by pruning and contraction produce more significant savings. There is evidence that problems for which the original BDD is intractable may often be easily analyzed using sound BDDs.

If a valid bound on the optimal value is available, cost-bounded BDDs could be a competitive method for solution as well as postoptimality analysis, particularly when the problem is nonlinear. This is a topic for future research.

Table 2. Experimental results for 0-1 MIPLIB instances with $\Delta = 0$.

instance	$\lvert B_\Delta \rvert$	$\lvert B_{cx \leq b} \rvert$	$\lvert B \rvert$
lseu	19	99	-
p0033	21	41	375
p0201	84	737	310420
stein27	4882	6260	25202
stein45	1176	1765	5102257

Acknowledgments

We would like to thank the anonymous reviewers for their valuable suggestions.

References

1. Hadzic, T., Hooker, J.: Postoptimality analysis for integer programming using binary decision diagrams. Technical report, Carnegie Mellon University (2006) Presented at GICOLAG workshop (Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geometry), Vienna.
2. Andersen, H.R.: An introduction to binary decision diagrams. Lecture notes, available online, IT University of Copenhagen (1997)
3. Barth, P.: Logic-based 0-1 Constraint Solving in Constraint Logic Programming. Kluwer, Dordrecht (1995)
4. Akers, S.B.: Binary decision diagrams. IEEE Transactions on Computers **C-27** (1978) 509–516
5. Lee, C.Y.: Representation of switching circuits by binary-decision programs. Bell Systems Technical Journal **38** (1959) 985?–999
6. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers **C-35** (1986) 677–?691
7. Bryant, R.E.: Symbolic boolean manipulation with ordered binary decision diagrams. ACM Computing Surveys **24** (1992) 293– 318
8. Becker, Behle, Eisenbrand, Wimmer: BDDs in a branch and cut framework. In: International Workshop on Experimental and Efficient Algorithms (WEA), LNCS. Volume 4. (2005)
9. Hadzic, T., Andersen, H.R.: A BDD-based Polytime Algorithm for Cost-Bounded Interactive Configuration. In: AAAI-Press. (2006)
10. Hosaka, K., Takenaga, Y., Kaneda, T., Yajima, S.: Size of ordered binary decision diagrams representing threshold functions. Theoretical Computer Science **180** (1997) 47–60
11. Jensen, R.M.: CLab: A C++ library for fast backtrack-free interactive product configuration. `http://www.itu.dk/people/rmj/clab/` (2007)
12. Lind-Nielsen, J.: BuDDy - A Binary Decision Diagram Package. `http://sourceforge.net/projects/buddy` (online)