Noname manuscript No.
(will be inserted by the editor)

Compact Representation of Near-Optimal Integer Programming Solutions

Thiago Serra · J. N. Hooker

May 2017, revised September 2018 and March 2019

Abstract It is often useful in practice to explore near-optimal solutions of an integer programming problem. We show how all solutions within a given tolerance of the optimal value can be efficiently and compactly represented in a weighted decision diagram. The structure of the diagram facilitates rapid processing of a wide range of queries about the near-optimal solution space, as well as reoptimization after changes in the objective function. We also exploit the paradoxical fact that the diagram can be reduced in size if it is allowed to represent additional solutions. We show that a "sound reduction" operation, applied repeatedly, yields the smallest such diagram that is suitable for postoptimality analysis, and one that is typically far smaller than a tree that represents the same set of near-optimal solutions. We conclude that postoptimality analysis based on sound-reduced diagrams has the potential to extract significantly more useful information from an integer programming model than was previously feasible.

Keywords decision diagrams · integer programming · postoptimality

1 Introduction

An integer programming model contains a wealth of information about the phenomenon it represents. An optimal solution of the model, or even a set of optimal solutions, captures only a small portion of this information. In many applications, it is useful to probe the model more deeply to explore

Thiago Serra
Carnegie Mellon University, Pittsburgh, USA
E-mail: tserra@cmu.edu

J. N. Hooker
Carnegie Mellon University, Pittsburgh, USA
E-mail: jh38@andrew.cmu.edu

alternative solutions, particularly solutions that are suboptimal as measured by the objective function but attractive for other reasons.

For example, in a recent study [10] an integer programming (IP) model was formulated to relocate distribution centers across Europe. In the absence of reliable estimates for fixed costs, the client opted for a suboptimal solution that relocated one rather than three distribution centers for only a 0.4% increase over the optimal cost. One may also wish to know which decisions are invariant across all near-optimal solutions. This was a key question in a nature reserve planning study [4] that sought to identify areas that are critical to protect native species. In addition, there are applications that require the solution of minor variations of a problem. In some combinatorial auctions [20], for example, a winners determination problem is first solved to maximize the sum of winning bids, and then re-solved with each winner removed by fixing certain variables to zero.

In general, one may wish to know which solutions are optimal or near-optimal when certain variables are fixed to desired values, or which values a given variable can take without sacrificing near-optimality. We may also wish to determine how much a cost coefficient can be perturbed without changing the optimal cost more than a certain amount.

These questions can be answered if the space of near-optimal solutions is compactly represented in a transparent data structure; that is, a data structure that can be efficiently queried to find near-optimal (or optimal) solutions that satisfy desired properties. In fact, the task of solving an IP model can be more generally conceived as the process of transforming an opaque data structure to a transparent data structure. The constraint set and objective function comprise an opaque structure that defines the problem but does not make good solutions apparent. A conventional solver transforms the problem statement into a very simple transparent structure: an explicit list of one or more optimal solutions. The ideal would be to derive a more general data structure that compactly but transparently represents the space of near-optimal solutions and how they relate to each other.

We propose a *weighted decision diagram* for this purpose. Binary and multivalued decision diagrams have long been used for circuit design, formal verification, and other purposes [2,9,29,34,38], but they can also compactly represent solutions of a discrete optimization problem [3,7,22,26]. A *weighted* decision diagram represents the objective function values as well. Such a diagram can be built to represent only near-optimal solutions, and it can can be easily queried for solutions that satisfy desired properties. This is because solutions correspond straightforwardly to paths in the diagram, and their objective function values to the length of the paths. To simplify exposition we restrict ourselves to 0–1 problems, but the algorithm for general integers is almost the same and requires only a different data structure to represent a multivalued rather than a binary decision diagram.

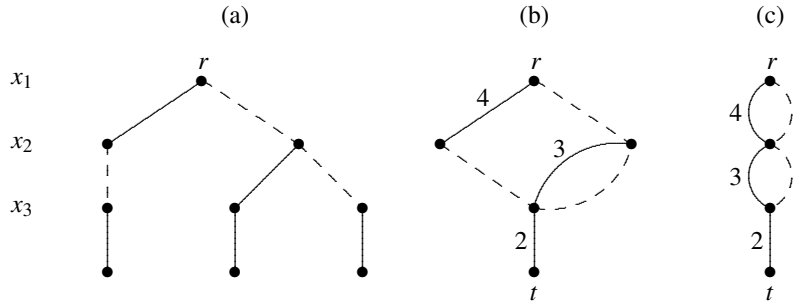

Fig. 1: (a) Branching tree for near-optimal solutions of (1) with $\Delta = 4$. (b) Reduced weighted decision diagram representing the same solutions. (c) Sound decision diagram for (1).

A simple example illustrates the idea. The IP problem

$$\begin{array}{ll} \text{minimize} & 4x_1 + 3x_2 + 2x_3 \\ \text{subject to} & x_1 + x_3 \geq 1, \;\; x_2 + x_3 \geq 1, \;\; x_1 + x_2 + x_3 \leq 2 \\ & x_1, x_2, x_3 \in \{0, 1\} \end{array} \tag{1}$$

has optimal value $z^* = 2$. The user wishes to explore solutions within $\Delta = 4$ of the optimum; that is, solutions with cost at most $z^* + \Delta$. Fig. 1(a) represents the solutions of interest, namely $(x_1, x_2, x_3) = (1, 0, 1)$, $(0, 1, 1)$, $(0, 0, 1)$. A dashed arc represents setting $x_j = 0$, and a solid arc represents setting $x_j = 1$. Figure 1(b) is a decision diagram that represents the same solutions. The solid arcs are assigned weights (lengths) equal to the corresponding objective function coefficients, while the dashed arcs have length zero. Each path from the root r to the terminus t represents a feasible solution with cost at most 6, where the cost of the solution is the length of the path.

The decision diagram of Fig. 1(b) is *reduced*, meaning that it is the smallest diagram that represents this set of solutions. It is well known that, for a given ordering of the variables, there is a unique reduced diagram representing a given set of solutions [9]. A diagram can be easily reduced with a single bottom-up pass that identifies and merges nodes with the same completions, as described in [9]. The ordering of the variables can sometimes drastically affect the size of the reduced diagram [5,6,8,13,25,28], but in this study we make no attempt to reorder variables to achieve smaller diagrams.

Although reduced decision diagrams tend to provide a much more compact representation than a branching tree, they can nonetheless grow rapidly. To address this issue, we take advantage of the fact that modifying a diagram to represent a larger solution set can, paradoxically, result in a smaller diagram. We adopt the concept of a *sound* decision diagram, introduced by [24]. This is a diagram that represents all near-optimal solutions along with some *spurious* solutions whose objective function values are worse than near-optimal (that is, greater than $z^* + \Delta$ when minimizing). By judiciously admitting spurious solutions into the diagram, we can significantly reduce its size while maintaining soundness.

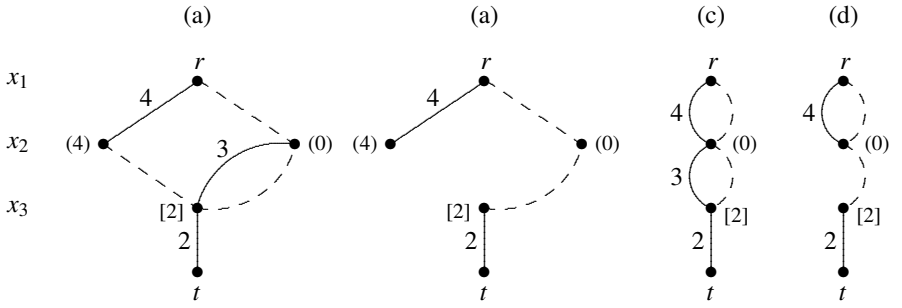

Fig. 2: (a) Reduced diagram for near-optimal solutions of (1), showing shortest path lengths from r to nodes in layer 2 and from the node in layer 3 to t. (b) Diagram showing possible values of x_2 in solutions within $\delta = 2$ of the optimum. (c) Sound reduced decision diagram for (1) showing shortest path lengths. (d) Diagram showing possible values of x_2 in solutions within $\delta = 2$ of the optimum.

Spurious solutions may be feasible or infeasible, because in either case they can be readily identified based on the fact that their cost is greater than $z^* + \Delta$. No time-consuming feasibility check is required, either during diagram construction or during postoptimality analysis. In fact, we will find that for many types of postoptimality analysis, it is not even necessary to recognize spurious solutions. It is enough that all near-optimal solutions be present in the diagram.

To create sound diagrams we employ a *sound reduction* operation that replaces a pair of nodes with a single node and yields a smaller sound diagram. Our main theoretical result is that repeated application of sound reduction operations, in any sequence, results in a smallest possible sound diagram for a given problem and variable ordering. It is smallest in the sense that it has a minimum number of arcs and a minimum number of nodes. We say that a diagram obtained in this fashion is *sound reduced*. Applying sound reduction operations to nodes in different orders can result in different sound-reduced diagrams, but remarkably, they all have the same minimum size.

Figure 1(c) illustrates a sound reduced diagram for problem (1). It represents the three solutions within $\Delta = 4$ of the optimal value $z^* = 2$, plus a spurious solution $(x_1, x_2, x_3) = (1, 1, 1)$ that is recognizable as such because its value is greater than $z^* + \Delta$. This solution happens to be infeasible, but its feasibility status is irrelevant.

We can illustrate a simple instance of postoptimality analysis using the diagrams of Fig. 1. Suppose the user wishes to know the possible values of x_2 in solutions within $\delta = 2$ of the optimal value. We first perform the analysis using the reduced diagram of Fig. 1(b), and then using the sound reduced diagram of Fig. 1(c). All the solutions of interest appear in these diagrams because the diagrams were created for $\Delta = 4$, and because $\delta \leq 4$.

In the reduced diagram of Fig. 1(b), we first compute the length of the shortest path from the root to each node in layer 2, shown in parentheses in Fig. 2(a). We also compute the longest path from the node in layer 3 to the

terminus, shown in brackets. Then two arcs leaving layer 2 can be removed as shown in Fig. 2(b) because no r–t path of length at most $z^* + \delta = 4$ can contain these arcs. This is evident from the shortest path lengths ($4+0+2 > 4$ and $0 + 3 + 2 > 4$). Since only a dashed arc now leaves layer 2, we conclude that x_2 can take only the value 0 in solutions within $\delta = 2$ of the optimum.

The same conclusion can be reached using the sound reduced diagram of Fig. 1(c). Shortest path lengths are calculated as shown in Fig. 2(c), and we see that no path of length at most $z^* + \delta = 4$ can contain the solid arc leaving layer 2. This yields the diagram of Fig. 2(d), and we again infer that $x_2 = 0$ in all solutions within $\delta = 2$ of the optimum. The spurious solution has no effect on the process because it is automatically screened out. This is only one type of postoptimality analysis; others are described after the theoretical development to follow.

We begin below with a review of related work, followed by four sections that develop the underlying theory of sound diagrams. Section 3 introduces some basic concepts and properties of decision diagrams. Section 4 develops the idea of soundness and shows that it is a useful concept only when suboptimal (as well as optimal) solutions are represented. Section 5 proves the main result that sound reduction yields a sound diagram of minimum size. It also shows by counterexample that there need not be a unique sound-reduced diagram for a given problem. Section 6 explains why it is not practical to admit superoptimal solutions into sound diagrams, even though this may result in smaller diagrams.

The remaining sections apply the theory of sound diagrams to integer programming. Section 7 presents an algorithm that constructs a sound diagram for a given integer programming problem, assuming that the optimal value has been obtained by solving the problem with a conventional solver. Section 8 shows how to introduce sound reduction into the algorithm, thereby obtaining a smallest possible sound diagram for the problem. Section 9 then describes several types of postoptimality analysis that can efficiently be performed on a sound diagram.

Section 10 reports computational tests that measure how compactly sound diagrams can represent near-optimal solutions, and the time required to compute the diagrams. Based on instances from MIPLIB, it is found that decision diagrams represent near-optimal solutions much more compactly than a branching tree, and that in most instances, sound reduction substantially reduces the size of the diagrams. In addition, the computational cost of constructing a sound diagram is generally not much greater than that of generating the near-optimal solutions it represents, and often much less. The paper concludes with a summary and agenda for future research.

2 Related Work

To our knowledge, no previous study addresses the issue of how to represent near-optimal solutions of IP problems in a compact and transparent fashion. A

few papers have proposed methods for *generating* multiple solutions. Scatter search is used in [19] to generate a set of diverse optimal and near-optimal solutions of mixed integer programming (MIP) problems. However, since it is a heuristic method, it does not obtain an exhaustive set of solutions for any given optimality tolerance.

Diverse solutions of an MIP problem have also been obtained by solving a sequence of MIP models, beginning with the given problem, in which each seeks a solution different from the previous ones. This approach is investigated in [21], where it is compared with solving a much larger model that obtains multiple solutions simultaneously. However, neither method is scalable, as there may be a very large number of near-optimal solutions.

A "branch-and-count" method is given in [1] for generating all *feasible* solutions of an IP problem, based on the identification of "unrestricted subtrees" of the branching tree. These are subtrees in which all values of the unfixed variables are feasible. We use a similar device, among others, when constructing sound decision diagrams. However, [1] presents no method for compactly representing solutions or organizing them in a way that is conducive to postoptimality analysis. In addition, the space of all feasible solutions is typically too large to analyze and of less interest to practitioners than near-optimal solutions.

The "one-tree" method of [11] generates a collection of optimal or near-optimal solutions of a mixed-integer programming problem by extending a branching tree that is used to solve the problem. The collection is not intended to be exhaustive, although parameters can be set to create an exhaustive list. There is no indication of how to represent the collection compactly or query it more easily. The commercial solver CPLEX, beginning with version 11.0, implements this method with a "solution pool" feature [31], which has been supported by the GAMS modeling system since version 22.6 [17].

Our work differs in that we focus on representing and analyzing near-optimal solutions rather than merely generating them. While we present an algorithm that generates solutions during construction of a sound diagram, we do so because (to our knowledge) only one solver has the capability to generate all near-optimal solutions. Our algorithm typically generates solutions more slowly than CPLEX, which is unsurprising, given that it lacks access to the cutting planes and other information gathered during the initial solution of the problem. Rather, it relies only on the optimal value, which can be obtained from any solver. The algorithm is easily modified to accept near-optimal solutions from CPLEX rather than generate them, and we report the resulting computational performance alongside that of the original algorithm.

Sound reduction is related to the analysis of dominance relations among search nodes while solving a problem [30, 33], where one wants to determine if the completions of a given node are contained in the completions of another. That can be particularly useful to correct branching decisions at the top of the tree, for example by checking if a different choice of values for the first variables may incur a smaller cost while defining a subproblem with at least the same completions as the node being explored. In the case, one has identified a node

that dominates the current one, and suboptimal solutions can be pruned [14, 15]. This approach is not suitable for our project, however, because we wish to preserve some of the suboptimal solution.

Integer programming sensitivity analysis has been investigated for some time, as for example in [12,16,18,27,32,36,37]. Sound decision diagrams can be used to analyze sensitivity to perturbations in objective coefficients, because these appear as arc lengths in the diagram, and we show how to do so. However, our main interest here is in probing the near-optimal solution set that results from the original problem data.

Decision diagrams were first proposed for IP postoptimality analysis in [23], and the concept of a sound diagram was introduced in [24]. The present paper extends this work in several ways. It proves several properties of sound diagrams, introduces the sound reduction operation, and proves that sound reduction yields a sound diagram of minimum size. It also presents algorithms for generating sound-reduced diagrams for IP problems and conducting postoptimality analysis on these diagrams, as well as reporting computational tests on the representational efficiency of the diagrams.

3 Decision Diagrams for Discrete Optimization Problems

For our purposes, we associate a decision diagram with a discrete optimization problem of the form

$$\min\{f(x) \mid x \in S\} \tag{P}$$

where $S \subseteq S_1 \times \ldots \times S_n$ and each variable domain S_j is finite. A *decision diagram* associated with (P) is a multigraph $D = (U, A, \ell)$ with the following properties:

- The node set U is partitioned $U = U_1 \cup \cdots \cup U_{n+1}$, where $U_1 = \{r\}$ and $U_{n+1} = \{t\}$. We say r is the *root node*, t the *terminal node*, and U_j is *layer* j of D for each j.
- The arc set A is partitioned $A = A_1 \cup \cdots \cup A_n$, where each arc in A_j connects a node in U_j with a node in U_{j+1}, for $j = 1, \ldots, n$.
- Each arc $a \in A_j$ has a *label* $\ell(a) \in S_j$ for $j = 1, \ldots, n$, representing a value assigned to variable x_j. The arcs leaving a given node must have distinct labels.

The labels on each path p of D from r to t represent an assignment to x, which we denote $x(p)$. We let $\mathrm{Sol}(D)$ denote the set of solutions represented by the r–t paths. We say that D *exactly represents* S when $\mathrm{Sol}(D) = S$.

A *weighted decision diagram* associated with (P) is a multigraph $D(U, A, \ell, w)$ that satisfies the above properties, plus the following:

- Each arc $a \in A$ has a *weight* $w(a)$, such that $\sum_{a \in p} w(a) = f(x(p))$ for any r–t path p of D. Thus the total weight $w(p)$ of an r–t path p is the objective function value of the corresponding solution.

Table 1: List of symbols.

r	root node of a decision diagram
t	terminal node of a decision diagram
U_j	set of nodes in layer j of a diagram
A_j	set of arcs connecting nodes in U_j with nodes in U_{j+1}
$\ell(a)$	label of arc a, representing value of x_j if $a \in A_j$
$w(a)$	weight (cost, length) of arc a
$x(p)$	assignment to x represented by r–t path p
$w(p)$	weight of r–t path p
$x(\pi)$	assignment to $x_j, \ldots, x_{k-1}$ represented by u–v path π ($u \in U_j$, v$\in U_k$)
$x_i(\pi)$	assignment to x_i represented by π, where $j \leq i < k$
$w(\pi)$	weight of path π
$w(u, u')$	weight of minimum-weight path from node u to node u'
$\text{Sol}(D)$	set of solutions represented by r–t paths in diagram D
z^*	optimal value of problem (P)
$\text{P}(\Delta)$	problem of finding Δ-optimal solutions of (P)
$S(\Delta)$	set of Δ-optimal solutions of (P)
$\text{ILP}(\Delta)$	problem of finding Δ-optimal solutions of (ILP)
$\text{Pre}(u)$	set of prefixes of node u
$\text{Suf}(u)$	set of suffixes of node u
$\text{Suf}_\Delta(u)$	set of Δ-suffices of node u of a diagram D; i.e., set of suffixes of u that are part of some Δ-optimal solution represented by D
$\text{lhs}.u$	left-hand-side state at node u
$\text{LCDS}_j[u, v]$	weight of least-cost differing suffix when reducing u into v
W	maximum width of (number of nodes in) layers of a diagram
$S_{\max}$	size of largest variable domain

A weighted decision diagram associated with problem (P) exactly represents (P) when $\text{Sol}(D) = S$. In this case, the optimal value z^* of (P) is the weight of any minimum-weight r–t path of D, and the optimal solutions of (P) are those corresponding to minimum-weight r–t paths. From here out, we will refer to a weighted decision diagram simply as a decision diagram, and to a diagram without weights as an unweighted decision diagram.

An unweighted decision diagram D is *reduced* when redundancy is removed. To make this precise, let a *suffix* of $u \in U_j$ be any assignment to $x_j, \ldots, x_n$ represented by a u–t path in D, and let $\text{Suf}(u)$ be the set of suffixes of u. Then D is reduced when $\text{Suf}(u) \neq \text{Suf}(v)$ for all $u, v \in U_j$ with $u \neq v$ and all $j = 1, \ldots, n$. As noted earlier, for any fixed variable ordering, there is a unique reduced unweighted decision diagram that exactly represents a given feasible set S, and this diagram is the smallest one that exactly represents S [9].

Given a path π from a node in layer j to a node in layer k, it will be convenient to denote by $x(\pi)$ the assignment to $(x_j, \ldots, x_{k-1})$ indicated by the labels on path π. We also let $x_i(\pi)$ denote the the assignment to x_i in particular, and we let $w(\pi)$ denote the weight of π. A summary of notation used throughout the paper can be found in Table 1.

The following simple property of decision diagrams will be useful.

Lemma 1 *Given any pair of distinct nodes u, v in layer j of a decision diagram, let π be an r–u path and ρ an r–v path. Then $x(\pi) \neq x(\rho)$.*

Proof If $x(\pi) = x(\rho)$, then in particular $x_1(\pi) = x_1(\rho)$. This implies that π and ρ lead from r to the same node u in U_2, since distinct arcs leaving r must have distinct labels. Arguing inductively, π and ρ lead from the same node in U_k to the same node in U_{k+1} for $k = 1, \ldots, j-1$. This implies that $u = v$, contrary to hypothesis. □

4 Sound Decision Diagrams

We are interested in constructing decision diagrams that represent near-optimal solutions of (P). By a near-optimal solution, we mean a feasible solution that is within a specified tolerance Δ of the optimum. We refer to such a solution x as Δ-*optimal*, meaning that $x \in S$ and $f(x) \leq z^* + \Delta$, for $\Delta \geq 0$. We denote by $S(\Delta)$ is the set of Δ-optimal solutions of (P), so that $S(0)$ is the set of optimal solutions. We let P(Δ) denote the problem of finding Δ-optimal solutions of (P).

We say that D exactly represents P(Δ) when $\mathrm{Sol}(D) = S(\Delta)$. Since such a decision diagram can be quite large, we wish to identify smaller diagrams that represent all the solutions in $S(\Delta)$ and allow them to be easily explored and analyzed. We therefore study decision diagrams that are *sound* for P(Δ), which represent a superset of $S(\Delta)$. A diagram D is sound when

$$S(\Delta) = \mathrm{Sol}(D) \cap \{x \in S_1 \times \cdots \times S_n \mid f(x) \leq z^* + \Delta\}$$

Thus a sound diagram can represent, in addition to Δ-optimal solutions, feasible and infeasible solutions that are worse than Δ-optimal but that can be easily identified as those corresponding to paths costing more than $z^* + \Delta$. We refer to these as *spurious* solutions. A *proper* sound diagram represents a proper superset of the Δ-optimal solutions and therefore represents some spurious solutions.

We will later show how to obtain a smallest possible sound diagram that represents all the solutions in S(Δ). To begin with, we prefer a sound diagram D that is *minimal* for P(Δ), meaning that every node of D, and every arc of D, lies on some r–t path that represents a solution in $S(\Delta)$. If a sound diagram is not minimal, one can easily determine which nodes and/or arcs are not minimal and those can be removed without destroying soundness. Since their removal does not enlarge the set represented by the diagram, we obtain a smaller diagram that is an equally accurate approximation of $S(\Delta)$.

Shortest path computations allow us to check when a node or arc can be removed while preserving soundness. For any two nodes u, u' in different layers of D, let $w(u, u')$ be the weight of a minimum-weight path from u to u' (infinite if there is no path). Then node u can be removed if and only if

$$w(r, u) + w(u, t) > z^* + \Delta$$

An arc a connecting $u \in U_j$ with $u' \in U_{j+1}$ can be removed if and only if

$$w(r, u) + w(a) + w(u', t) > z^* + \Delta \tag{2}$$

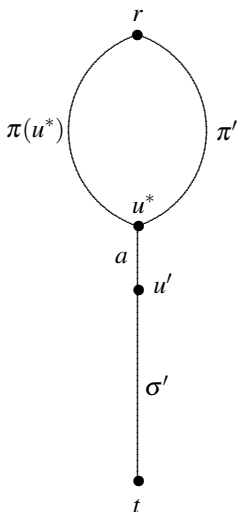

Fig. 3: Illustration of the proof of Theorem 1.

Interestingly, a proper sound diagram for P(0) is never minimal. This implies that there is no point in considering sound diagrams to represent the set of optimal solutions. They are useful only for representing sets of near-optimal solutions.

Theorem 1 *No proper sound decision diagram is minimal for* $P(0)$.

Proof Suppose to the contrary that diagram D is a minimal for P(0) and contains a suboptimal r–t path p. For any given node u in p, let $\pi(u)$ be the portion of p from r to u. Select a node u^* in p that maximizes the number of arcs in $\pi(u^*)$ subject to the condition that $\pi(u^*)$ is part of some optimal (minimum-weight) r–t path in D (Fig. 3). We note that $u^* \notin U_{n+1}$, since otherwise p would be an optimal r–t path. Thus p contains an arc a from u^* to some node u'. Furthermore, $u^* \notin U_1$ since otherwise arc a would prevent D from being minimal. Now since D is minimal, arc a belongs to some optimal r–t path, which we may suppose consists of π', a, and σ'. Hence both $\pi(u^*)$ and π' are part of optimal r–t paths, which implies that both are optimal r–u^* paths. It follows that the r–t path consisting of $\pi(u')$ and σ' is also optimal. This implies that $\pi(u')$, which contains one more arc than $\pi(u^*)$, is part of an optimal r–t path, contrary to the definition of u^*. □

The following property of sound diagrams is easily verified.

Lemma 2 *If a decision diagram* D *is sound for* $P(\Delta)$*, then* D *is sound for* $P(\delta)$ *for any* $\delta \in [0, \Delta]$.

Thus the set of sound decision diagrams of $P(\Delta)$ is a subset of that of $P(\delta)$ for any $\delta \in [0, \Delta]$.

Corollary 1 *The size of a smallest sound diagram for $P(\Delta)$, as measured by the number of arcs or the number of nodes, is monotone nondecreasing in Δ.*

5 Sound Reduction

Sound reduction is a tool for reducing the size of a given sound diagram, generally at the cost of increasing the number of spurious solutions it represents. Given distinct nodes $u, v \in U_j$ for $1 < j \leq n$, we can *sound-reduce u into v* when diverting to v the arcs coming into u, and deleting u from the diagram, removes no Δ-optimal solutions and adds only spurious solutions. Thus sound reduction removes at least one node without destroying soundness. In fact, we will see that repeated sound reduction yields the smallest sound diagram for a given Δ.

Let a *Δ-suffix* of node $u \in U_j$ be any suffix in $\mathrm{Suf}(u)$ that is part of a Δ-optimal solution, and let $\mathrm{Suf}_\Delta(u)$ be the set of Δ suffixes of u. Also let a *prefix* of u be any assignment to $(x_1, \ldots, x_{j-1})$ represented by an r–u path, and let $\mathrm{Pre}(u)$ be the set of prefixes of u. Then u can be sound-reduced into v if:

$$\mathrm{Suf}_\Delta(u) \subseteq \mathrm{Suf}(v) \tag{3}$$

$$w(\pi) + w(\sigma) > z^* + \Delta \text{ when } x(\pi) \in \mathrm{Pre}(u) \text{ and } x(\sigma) \in \mathrm{Suf}(v) \setminus \mathrm{Suf}(u) \tag{4}$$

Sound reduction is accomplished as follows. For every arc a from some node $q \in U_{j-1}$ to u, remove a and create an arc from q to v with label $\ell(a)$ and weight $w(a)$. Then remove u and any successor of u that is disconnected from r. That is, remove u and any successor u' of u for which all r–u' paths in D contain u.

Condition (3) ensures that any Δ-optimal solution whose r–t path passes through u remains in the diagram after sound reduction, with the same cost. Condition (4) ensures that only spurious solutions are added to the diagram. So we have,

Theorem 2 *Sound reduction preserves soundness.*

Figure 4 illustrates sound reduction. Figure 4(a) is a reduced diagram that is sound for a problem $\mathrm{P}(\Delta)$ with $z^* = 2$ and $\Delta = 6$. Dashed arcs have label 0 and weight 0, and solid arcs have label 1 and weights as shown. Figure 4(b) shows the result of sound-reducing node u_1 into node v_1. Condition (3) is satisfied because $\mathrm{Suf}_\Delta(u_1) = \{(1,1,0,0)\} \subseteq \{(1,1,0,0),(1,1,0,1)\} = \mathrm{Suf}(v_1)$. Condition (4) is satisfied because $\mathrm{Pre}(u_1) = \{(1,1)\}$, $\mathrm{Suf}(v_1) \setminus \mathrm{Suf}(u_1) = \{(1,1,0,1)\}$, and the solution $(x_1, \ldots, x_6) = (1,1,1,1,0,1)$ has cost $9 > z^* + \Delta$. We could have also reduced u_2 into v_2, u_3 into v_3, or u_3 into q.

A sound diagram for $\mathrm{P}(\Delta)$ is *sound-reduced* if no further sound reductions are possible. We can show that a minimal sound-reduced diagram is the smallest diagram that is sound for $\mathrm{P}(\Delta)$. For example, the diagram in Fig. 4(b) is sound-reduced, and it is in fact the smallest sound diagram for $\mathrm{P}(\Delta)$ with $\Delta = 6$. Establishing this result requires two lemmas.

Lemma 3 *Given a sound-reduced diagram D for $P(\Delta)$, any two distinct nodes $u, v \in U_j$ of D satisfy* $\text{Suf}_\Delta(u) \neq \text{Suf}_\Delta(v)$.

Proof Suppose to the contrary that $\text{Suf}_\Delta(u) = \text{Suf}_\Delta(v)$, and assume without loss of generality that $w(r,u) \geq w(r,v)$. We will show that u can be sound-reduced into v, contrary to hypothesis. Condition (3) for sound reduction is obviously satisfied. Also condition (4) is satisfied, because if $x(\pi) \in \text{Pre}(u)$ and $x(\sigma) \in \text{Suf}(v) \setminus \text{Suf}(u)$, then $x(\sigma) \notin \text{Suf}_\Delta(u)$, and therefore $x(\sigma) \notin \text{Suf}_\Delta(v)$. This implies $w(r,v) + w(\sigma) > z^* + \Delta$. But $w(\pi) + w(\sigma) \geq w(r,u) + w(\sigma) \geq w(r,v) + w(\sigma)$, and (4) follows. □

Lemma 4 *Let D be a minimal sound-reduced diagram for $P(\Delta)$. For any node u in layer j of D, and any other diagram D' with the same variable ordering that is sound for $P(\Delta)$, there is a node u' in layer j of D' with* $\text{Suf}_\Delta(u) = \text{Suf}_\Delta(u')$.

Proof Suppose to the contrary that there is a node u in layer j of D, and some sound diagram D' in which layer j contains no node with the same Δ-suffixes as u. We will show that D must then contain a node v into which u can be sound-reduced, contrary to hypothesis.

Let π be a minimum-weight r–u path in D. Since D is minimal, node u belongs to some path that represents a Δ-optimal solution. So since π is a minimum-weight r–u path, $x(\pi)$ is the prefix of some Δ-optimal solution. Thus since D' is sound for $\text{P}(\Delta)$, layer j of D' must contain a node u' and an r–u' path π' with $x(\pi') = x(\pi)$. Since every Δ-optimal solution represented by D is also represented by D', we have that $\text{Suf}_\Delta(u) \subseteq \text{Suf}_\Delta(u')$, due to the fact that π is a minimum-weight path. However, by hypothesis $\text{Suf}_\Delta(u) \neq \text{Suf}_\Delta(u')$, and so we have $\text{Suf}_\Delta(u') \setminus \text{Suf}_\Delta(u) \neq \emptyset$.

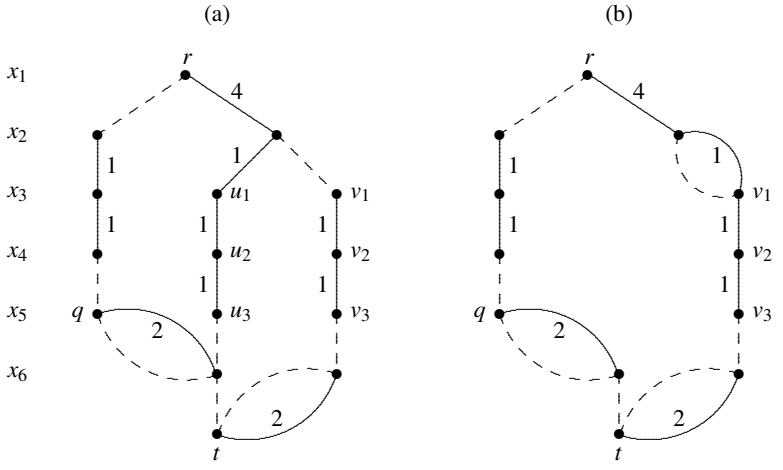

Fig. 4: Reduced (a) and sound-reduced (b) decision diagrams for $z^* = 2$ and $\Delta = 6$.

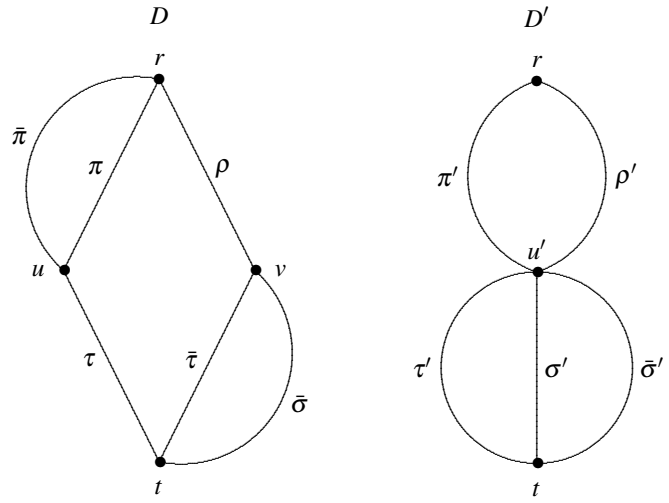

Fig. 5: Illustration of the proof of Lemma 4.

Now consider any u'–t path σ' for which $x(\sigma') \in \mathrm{Suf}_{\Delta}(u') \setminus \mathrm{Suf}_{\Delta}(u)$. This implies that $x(\sigma')$ is the suffix of some Δ-optimal solution, and so there must be an r–u' path ρ' with

$$w(\rho') + w(\sigma') \leq z^* + \Delta \tag{5}$$

However, we can see as follows that $(x(\pi'), x(\sigma'))$ is not a Δ-optimal solution. Note that by Lemma 1, π is the only path in D representing $x(\pi) = x(\pi')$. Thus if $(x(\pi'), x(\sigma'))$ were Δ-optimal, the soundness of D would imply that $x(\sigma') = x(\sigma) \in \mathrm{Suf}_{\Delta}(u)$ for some u–t path σ, which contradicts the fact that $x(\sigma') \in \mathrm{Suf}_{\Delta}(u') \setminus \mathrm{Suf}_{\Delta}(u)$. So $(x(\pi'), x(\sigma'))$ is not Δ-optimal, which means $w(\pi') + w(\sigma') > z^* + \Delta$. This and (5) imply $w(\rho') < w(\pi')$. But (5) also implies that the sound diagram D must contain a node v and an r–v path ρ with $x(\rho') = x(\rho)$, so that $w(\rho) < w(\pi)$. Since π is a minimum-weight r–u path, this implies $u \neq v$.

We now show that u can be sound-reduced into v by verifying conditions (3) and (4). To show (3), consider any u–t path τ with $x(\tau) \in \mathrm{Suf}_{\Delta}(u)$. Since π is a minimum-weight r–u path, $(x(\pi), x(\tau))$ is a Δ-optimal solution. Now since D' is sound for $\mathrm{P}(\Delta)$ and $x(\pi) = x(\pi')$, there is a u'–t path τ' in D' for which $(x(\pi'), x(\tau'))$ is Δ-optimal. This means $(x(\rho'), x(\tau'))$ is Δ-optimal because $w(\rho') < w(\pi')$, which implies that $(x(\rho), x(\tau))$ is Δ-optimal. Since by Lemma 1, ρ is the only path representing $x(\rho)$, there must be a v–t path $\bar{\tau}$ with $x(\bar{\tau}) = x(\tau)$ and $x(\bar{\tau}) \in \mathrm{Suf}_{\Delta}(v)$. This implies $x(\tau) \in \mathrm{Suf}(v)$ and (3).

Finally, to show (4), let $\bar{\pi}$ be an r–u path with $x(\bar{\pi}) \in \mathrm{Pre}(u)$, and let $\bar{\sigma}$ be a v–t path with $x(\bar{\sigma}) \in \mathrm{Suf}(v) \setminus \mathrm{Suf}(u)$. Note that if $w(\rho) + w(\bar{\sigma}) > z^* + \Delta$, then since $w(\bar{\pi}) \geq w(\pi) > w(\rho)$, we have $w(\bar{\pi}) + w(\bar{\sigma}) > z^* + \Delta$, and (4) follows. We may therefore suppose $w(\rho) + w(\bar{\sigma}) \leq z^* + \Delta$, which means that $(x(\rho), x(\bar{\sigma}))$

is Δ-optimal because D is sound. Since D' is sound and $x(\rho) = x(\rho')$, by Lemma 1 there must be a u'–t path $\bar{\sigma}'$ for which $x(\bar{\sigma}') = x(\bar{\sigma})$ and $(x(\rho'), x(\bar{\sigma}'))$ is Δ-optimal. This means that D' represents the solution $(x(\pi'), x(\bar{\sigma}'))$, which is the same as $(x(\pi), x(\bar{\sigma}))$. But since $x(\bar{\sigma}) \notin \text{Suf}(u)$, D does not represent the solution $(x(\pi), x(\bar{\sigma}))$, which therefore cannot be Δ-optimal. Thus since D' represents this solution, it must be spurious, and we have $w(\pi)+w(\bar{\sigma}) > z^*+\Delta$. This implies $w(\bar{\pi}) + w(\bar{\sigma}) > z^* + \Delta$ and (4). □

Theorem 3 *A sound decision diagram D for $P(\Delta)$ has a minimum number of nodes and a minimum number of arcs, among diagrams that are sound for $P(\Delta)$ and have the same variable ordering, if and only if D is minimal and sound-reduced.*

Proof If D is not minimal, we can remove one or more nodes or arcs, and if D is not sound-reduced, we can remove at least one node. Thus D is minimal and sound-reduced if it has a minimum number of nodes and arcs.

To prove the converse, suppose D is minimal and sound-reduced. Due to Lemma 3, all nodes in any given layer j of D have sets of Δ-suffixes. By Lemma 4, these distinct sets of Δ-suffixes exist for nodes in layer j of any sound diagram for $\text{P}(\Delta)$. Thus any sound diagram for $\text{P}(\Delta)$ has at least as many nodes as D. Furthermore, the minimality of D implies that any arc a leaving a node u in layer j of D is part of some Δ-optimal solution. Given any diagram D' that is sound for $\text{P}(\Delta)$, the node u' in layer j of D' with $\text{Suf}_\Delta(u') = \text{Suf}_\Delta(u)$ must have an outgoing arc with the same label as a. Thus D' has at least as many arcs as D. □

Although all sound-reduced diagrams for a given $\text{P}(\Delta)$ have minimum size, they are not necessarily identical. For example, while all sequences of sound reductions of Fig. 4(a) terminate in the same diagram Fig. 4(b), this is not the case for the slightly different diagram of Fig. 6(a). Sound-reducing u_3 into q yields the sound-reduced diagram of Fig. 6(b), and sound-reducing u_3 into v_3 yields Fig. 6(c). Both diagrams are of minimum size and satisfy Lemma 4, but they are distinct.

When a node can be sound-reduced into more than one node, as in this example, one can choose the node with fewer prefixes and suffices. This tends to reduce the number of spurious solutions in the diagram.

6 Two-Sided Soundness

A sound diagram is permitted to represent feasible and infeasible solutions that are worse than Δ-optimal. A natural question is whether it would be useful to allow (infeasible) solutions that are *better than optimal*. Superoptimal solutions, like solutions that are worse than Δ-optimal, can be filtered out during postoptimality analysis by examining only objective function values.

Since such a diagram D requires excluding solutions with values on either side of the interval $[z^*, z^* + \Delta]$, we will say that it has *two-sided soundness*,

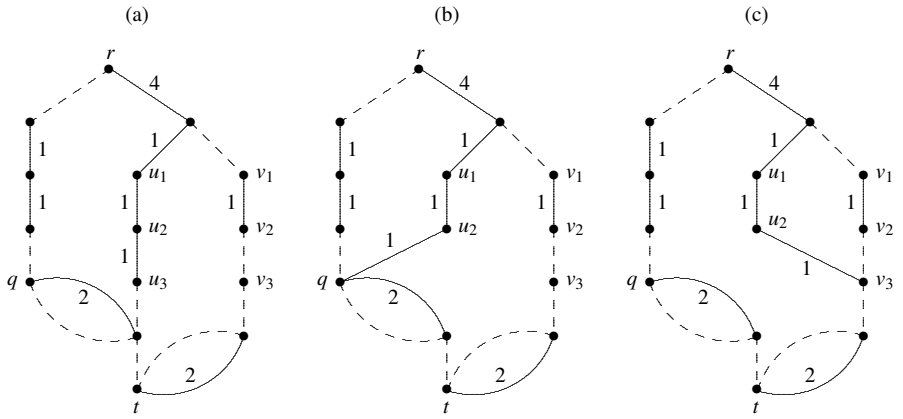

Fig. 6: Distinct sound-reduced diagrams (b) and (c) obtained from diagram (a), where $z^* = 2$ and $\Delta = 6$.

meaning that it satisfies

$$S(\Delta) = \mathrm{Sol}(D) \cap \{x \in S_1 \times \cdots \times S_n \mid z^* \leq f(x) \leq z^* + \Delta\}$$

Conceivably, this weaker condition for including solutions could allow more flexibility for finding a small sound diagram.

There is a theoretical reason, however, that two-sided soundness is less suitable for practical application. In a one-sided sound diagram, it is easy to check whether a given r–u path of cost w can be completed to represent a δ-optimal solution, which only depends on pre-computing shortest paths that can be later retrieved. Namely, find the value of a shortest u–t path and check whether its length is at most $z^* - w + \delta$. In a two-sided sound diagram, this decision problem is NP-complete. It is therefore difficult to extract δ-optimal solutions from a two-sided sound diagram.

Theorem 4 *Checking whether some r–t path in a given decision diagram has cost that lies in an given interval $[z^*, z^* + \delta]$ is NP-complete.*

Proof The problem belongs to NP because an r–t path with cost in $[z^*, z^* + \delta]$ is a polynomial-size certificate. It is NP-complete because we can reduce the subset sum problem to it. Given a set $S = \{s_1, \ldots, s_n\}$ of integers, the subset sum problem is to determine whether some nonempty subset of these integers sums to zero. We can solve the problem by constructing a decision diagram as follows (an example with $n = 3$ appears in Fig. 7). Let $U_1 = \{r\}$, $U_j = \{u_{j1}, \ldots, u_{jn}\}$ for $j = 2, \ldots, n$, and $U_{n+1} = \{t\}$. There are two arcs from each u_{jk} to $u_{j+1,k}$ for $j = 2, \ldots, n-1$, one with weight 0, and the other with weight s_j if $k < j$ and weight s_{j-1} otherwise. There are two arcs from each u_{nk} to t, one with weight zero, and the other with weight s_n if $k < n$ and weight s_{n-1} otherwise. Then all and only nonempty subsets of S are represented by r–t

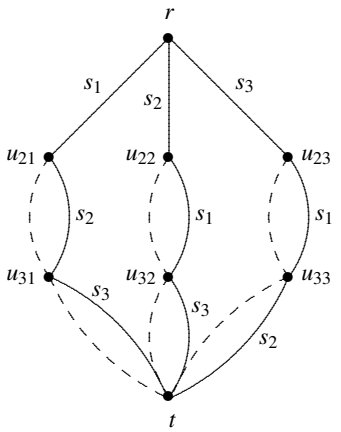

Fig. 7: Example of the decision diagram representing nonempty subsets of S for $|S| = 3$, as described in the proof of Theorem 4. A dashed line indicates that no element is selected.

paths of the diagram. The subset sum problem has a solution if and only if there is an r–t path with cost in the interval $[0, 0]$. □

Corollary 2 *Checking whether a given r–u path can be extended to a path with cost in $[z^*, z^* + \delta]$ is NP-complete.*

Proof Let the given r–u path in diagram D have cost w, and consider the decision diagram D' consisting of all u–t paths of D. The path extension problem is equivalent to checking whether some u–t path in D' has cost in the interval $[z^* - w, z^* - w + \delta]$, which by Theorem 4 is an NP-complete problem. □

7 Sound Diagrams for Bounded Integer Linear Programs

We now specialize problem (P) to an integer linear program:

$$\min\{cx \mid Ax \geq b,\ x \in S_1 \times \ldots \times S_n\} \tag{ILP}$$

in which A is an $m \times n$ matrix and each S_j is a finite set of integers. We wish to build a sound decision diagram that represents Δ-optimal solutions of (ILP); that is, a sound diagram for ILP(Δ). We assume that (ILP) has been solved to optimality and the optimal value z^* is known. This will accelerate the construction of a sound diagram.

We build the diagram by constructing a branching tree, identifying nodes that necessarily have the same set of Δ-suffixes, and removing some nodes that cannot be part of a Δ-optimal solution. To accomplish this, we associate with every node $u \in U_j$ a *state* $(u.\text{lhs}, w(r, u))$. In the simplest case, u.lhs is an

m-tuple in which component i is the sum of left-hand-side terms of inequality constraint i that have been fixed by branching down to layer j. The root node r initially has state $(\mathbf{0}, 0)$, where $\mathbf{0}$ is a tuple of zeros. We can identify nodes u, u' that have the same lhs state, because they necessarily have the same Δ-suffixes. The resulting node v has state $(v.\text{lhs}, w(r,v))$, where $v.\text{lhs} = u.\text{lhs}$ and $w(r,v) = \min\{w(r,u), w(r,u')\}$. Thus the state variable $w(r,v)$ maintains the weight of a minimum-weight r–v path in the current diagram.

We can also observe that inequality constraint i is satisfied at node $u \in U_j$, for any values of $x_j, \dots, x_n$, when the sum of the fixed terms on the left-hand side is sufficiently large. Specifically, inequality i is necessarily satisfied when the sum of these terms is at least $b_i - M_{ij}$, where

$$M_{ij} = \sum_{k=j}^{n} \min\left\{A_{ik}x_k \mid x_k \in S_k\right\}$$

This allows us to update the lhs state to $\min\{u.\text{lhs},\ b - M_j\}$ and still identify nodes that have the same state. Here, $M_j = (M_{1j}, \dots, M_{mj})$, and the minimum is taken componentwise.

We can remove a node $u \in U_j$ when the cost of any r–t path through u must be greater than $z^* + \Delta$, based on the linear relaxation of (ILP) at node u. We therefore remove u when $w(r,u) + \text{LP}_j(u.\text{lhs}) > z^* + \Delta$, where

$$\text{LP}_j(u.\text{lhs}) = \min\left\{ \sum_{k=1}^{j-1} c_k x_k \;\middle|\; \sum_{k=j}^{n} A_k x_k \geq b - u.\text{lhs},\ \ x_k \in I_k,\ k = j, \dots, n \right\}$$

and I_k is the interval $[\min S_k, \max S_k]$. This can remove some spurious solutions, but not necessarily all, because $w(r,u)$ can underestimate the weight of r–u paths, and $\text{LP}_j(u.\text{lhs})$ can underestimate the weight of u–t paths. One can understand this step as pruning node u because all solutions passing through it cost more than the near-optimal threshold, which may happen more often if a smaller Δ is used, to the point of removing all nodes that are not incident to shortest r–t paths when $\Delta = 0$.

The diagram construction is controlled by Algorithm 1, which maintains unexplored nodes in a priority queue that determines where to branch next. When exploring node $u \in U_j$, the procedure invokes Algorithm 2 to create an outgoing arc for each value in the domain S_j of x_j. Some of these arcs may lead to dead-end nodes based the LP relaxation as described above. If all lead to dead ends, u and predecessors of u with no outgoing arcs are removed by the subroutine at the bottom of Algorithm 1.

Each surviving arc a out of u is processed as follows. Let the node q at the other end of arc a have state $(q.\text{lhs}, w(r,u) + c_j\ell(a))$, where

$$q.\text{lhs} = \min\left\{b - M_j,\ u.\text{lhs} + A_j\ell(a)\right\}$$

If no node currently in U_{j+1} has the same lhs state as q, add node q to U_{j+1}. Otherwise, some node $v \in U_{j+1}$ has $v.\text{lhs} = q.\text{lhs}$, and we let arc a run from u

Algorithm 1 Builds sound diagram for (ILP) using an arbitrary search type

```
1: procedure PopulateSoundDiagram()
2:     r ← new Node(min{0, M_1}, 0)
3:     U_1 ← {r}
4:     PriorityQueue ← {(1, r)}                    ▷ Begins search at root node r
5:     RevisitBFSQueue ← {}
6:     while PriorityQueue ≠ ∅ do
7:         (j, u) ← PriorityQueue.pop()            ▷ Queue policy defines search type
8:         u.openArcs ← |S_j|
9:         Deadend ← true
10:        for α ∈ S_j do
11:            Success ← TryBranching(j, u, α)     ▷ Algorithm 2
12:            Deadend ← Deadend ∧ ¬Success
13:        end for
14:        if Deadend then                         ▷ No branch succeeded
15:            RemoveDeadendNode(j, u)             ▷ Procedure below
16:        else if RevisitBFSQueue ≠ ∅ then        ▷ Nodes following u reopened
17:            RevisitNodes()                      ▷ Algorithm 3
18:        else if u.openArcs = 0 then             ▷ Nodes following u are all closed
19:            CloseNode(j, u)                     ▷ Algorithm 4
20:        end if
21:        u.explored ← true
22:    end while
23: end procedure
```

Subroutine: Removes nodes that cannot reach t recursively

```
24: procedure RemoveDeadendNode(j, u)
25:     U_j ← U_j \ {u}
26:     for all a = (v, u) ∈ A do
27:         A ← A \ {a}
28:         v.openArcs ← v.openArcs − 1
29:         if ∄a' = (v, u') ∈ A : u' ≠ u then      ▷ Node above is a deadend
30:             RemoveDeadendNode(j − 1, v)
31:         else if v.openArcs = 0 then
32:             CloseNode(j − 1, v)
33:         end if
34:     end for
35: end procedure
```

to v, updating $w(r, v)$ if necessary. If v has been explored already, it is revisited in Algorithm 3, because the updated value of $w(r, v)$ may affect which nodes and arcs can be deleted.

A key concept in the procedure is that of a *closed* node. The terminal node t is designated as closed ($t.\texttt{closed} = \mathit{true}$) when it is first reached in Algorithm 2. Higher nodes in the diagram are recursively marked as closed when all of their successors are closed. The recursion is implemented by maintaining the number $u.\texttt{openArcs}$ of arcs from node u that do not lead to closed nodes. When Algorithm 1 pops u from the priority queue and processes it, $u.\texttt{openArcs}$ is set to the number $|S_j|$ of domain elements of x_j. Algorithm 1 decrements this number for each dead-end arc, and Algorithm 2 decrements it for each arc leading to a pre-existing node that is closed. Node u is closed when $u.\texttt{openNodes}$ reaches zero.

Algorithm 2 Tries to branch on value and creates a new node if needed

```
 1: function TryBranching(j, u, α)
 2:     nodeLhs ← min{u.lhs + A_j α, M_j}
 3:     nodeWeight ← w(r, u) + c_j α
 4:     if nodeWeight + LP_j(u.lhs) > z* + Δ then          ▷ LP = ∞ if infeasible
 5:         return false
 6:     end if
 7:     if ∃v ∈ U_{j+1} : v.lhs = nodeLhs then             ▷ Found node with same lhs
 8:         if nodeWeight < w(r, v) then                   ▷ Improves minimum cost path to r
 9:             w(r, v) ← nodeWeight
10:             if v.explored then
11:                 RevisitBFSQueue.add(j + 1, v)          ▷ For Algorithm 3
12:             end if
13:         end if
14:         A ← A ∪ {(u, v)}
15:         if v.closed then                               ▷ Fails if not improving for a closed node
16:             u.openArcs ← u.openArcs − 1
17:         end if
18:     else                                               ▷ Creates node for new lhs
19:         v ← new Node(nodeLhs, nodeWeight)
20:         U_{j+1} ← U_{j+1} ∪ {v}
21:         A ← A ∪ {(u, v)}
22:         if j < n then                                  ▷ Adds non-terminal node to queue
23:             PriorityQueue.add(j + 1, v)
24:         else                                           ▷ First reached terminal node t
25:             v.closed ← true
26:             w(v, t) ← 0
27:         end if
28:     end if
29:     return true
30: end function
```

One purpose of the node closing mechanism is to implement a possibly more effective test for removing nodes than the LP relaxation. When the terminal node t is reached, a third state variable $w(t,t)$ is set to 0. When a node u is closed, the state variable $w(u,t)$ is updated to indicate the weight of a minimum-weight path to t. Algorithm 4 then removes node u if $w(r,u) + w(u,t) > z^* + \Delta$. It also removes an outgoing arc a to a node v when $w(r,u) + c_j\ell(a) + w(v,t) > z^* + \Delta$. Even this test, however, may not remove all spurious solutions.

In essence, the construction described here consists of a generic branching algorithm for a fixed variable ordering, coupled with methods that determine when each node has been fully explored. The complexity of this procedure depends on that of enumerating near-optimal solutions by branch and bound.

8 Algorithm for Sound Reduction

Applying the conditions (3)–(4) for sound reduction presupposes that the suffixes of nodes u and v are known, as well as the weight of a minimum-weight path from v to the terminal node. Sound reduction is therefore attempted only

Algorithm 3 Revisits nodes already explored for updating and re-branching

```
 1: procedure RevisitNodes()
 2:     while RevisitBFSQueue ≠ ∅ do
 3:         (j, u) ← RevisitBFSQueue.pop()
 4:         if u.closed then
 5:             ReopenNode(u)                                  ▷ Procedure below
 6:         end if
 7:         for α ∈ S_j do
 8:             if ∄a = (u, v) ∈ A : l(a) = α then             ▷ Branches again on absent α
 9:                 u.openArcs ← u.openArcs + 1
10:                 TryBranching(j, u, α)                      ▷ Algorithm 2
11:             else
12:                 if w(r, u) + c_j α < w(r, v) then          ▷ Improves w(r, v)
13:                     w(r, v) ← w(r, u) + c_i α
14:                     if v.explored then
15:                         RevisitBFSQueue.add(j + 1, v)
16:                     end if
17:                 end if
18:             end if
19:         end for
20:         if u.openArcs = 0 then                             ▷ Nodes following u remained closed
21:             CloseNode(j, u)                                ▷ Algorithm 4
22:         end if
23:     end while
24: end procedure
```

Subroutine: Reopens nodes in bottom-up order recursively

```
25: procedure ReopenNode(u)
26:     u.closed ← false
27:     for all a = (v, u) ∈ A do                              ▷ Opens all nodes above
28:         if v.closed then
29:             ReopenNode(v)
30:             v.openArcs ← 1
31:         else
32:             v.openArcs ← v.openArcs + 1
33:         end if
34:     end for
35: end procedure
```

when a node is closed, because it is at this point that the necessary information becomes available.

Algorithm 5 attempts to sound-reduce u into other closed nodes in the same layer, and to sound-reduce other nodes in the layer into u. It is invoked at line 23 in Algorithm 4. To check the conditions for sound-reducing u into v, Algorithm 5 recursively computes the weight of a minimum-weight suffix of v that is not a suffix of u (and similarly with u and v reversed). We refer to this as a *least-cost differing suffix* (LCDS) and denote its weight by $\text{LCDS}_j[v, u]$. The computation of $\text{LCDS}_j[v, u]$ and $\text{LCDS}_j[u, v]$ occurs in lines 4–17 of the algorithm.

The test for sound-reducing u into v occurs in lines 18–22. To break symmetry, we attempt the sound-reduction only when $w(r, v) \leq w(r, u)$. A failure of condition (3) for sound reduction occurs when a Δ-suffix of u is not a suffix of v, so that $w(r, u) + \text{LCDS}_j[u, v] \leq z^* + \Delta$. Condition (4) is

Algorithm 4 Closes nodes and performs bottom-up processing recursively

```
 1: procedure CloseNode(j, u)
 2:     u.closed ← true
 3:     for all a = (u, v) ∈ A do                    ▷ Computes minimum cost path to t
 4:         w(u, t) ← min{w(u, t), c_j ℓ(a) + w(v, t)}
 5:     end for
 6:     if w(r, u) + w(u, t) > z* + Δ then                          ▷ Node is not minimal
 7:         RemoveDeadendNode(i, u)                          ▷ Subroutine in Algorithm 1
 8:     end if
 9:     for all a = (u, v) ∈ A do
10:         if w(r, u) + c_j ℓ(a) + w(u, t) > z* + Δ then           ▷ Arc is not minimal
11:             A ← A \ {a}
12:         end if
13:     end for
14:     ClosingQueue ← {}
15:     for all a = (v, u) ∈ A do
16:         if ¬v.closed then
17:             v.openArcs ← v.openArcs − 1
18:             if v.openArcs = 0 then                              ▷ Closes node above
19:                 ClosingQueue ← ClosingQueue ∪ {(j, u)}
20:             end if
21:         end if
22:     end for
23:     CompressDiagram(j, u)                              ▷ Algorithm ?? or 5
24:     while ClosingQueue ≠ ∅ do
25:         (j, u) ← ClosingQueue.pop()
26:         CloseNode(j, u)
27:     end while
28: end procedure
```

violated when v has a suffix that is not a suffix of u and incurs a cost no greater than $z^* + \Delta$ when combined with some prefix of u. This occurs when $w(r,u) + \mathrm{LCDS}_j[v,u] \leq z^* + \Delta$. We can therefore sound-reduce u into v when

$$w(r,u) + \min\left\{\mathrm{LCDS}_j[u,v], \mathrm{LCDS}_j[v,u]\right\} > z^* + \Delta$$

The analogous test for sound-reducing v into u occurs in lines 23–26. The removal of a node during sound reduction may disconnect subsequent nodes in the diagram, which are removed by the subroutine at the bottom of Algorithm 5.

Sound reduction is relatively efficient. Algorithm 5 is called $O(nW)$ times, where W is the maximum width of a layer. Each call checks $O(W)$ nodes having $O(S_{\max})$ arcs each, where $S_{\max}$ is the size of the largest variable domain. This totals $O(nW^2 S_{\max})$ operations before node removals. Each call of the bottom procedure requires time $O(S_{\max})$, for a total time of $O(nWS_{\max})$.

The sound reduction procedure of Algorithm 5 can substantially reduce the space requirements of postoptimality analysis, because it reduces the size of the resulting diagram. However, it can add significantly to the construction time for the diagram. The computational experiments reported in Section 10 illustrate the time/space tradeoff for a number of problem instances.

Algorithm 5 Sound-reduces node u with another closed node if possible

procedure CompressDiagram(j, u)
  for all $v \in U_j : v \neq u \wedge v$.closed, and v ordered by nondecreasing $w(r, d)$ **do**
    $\text{LCDS}_j[u, v], \text{LCDS}_j[v, u] \leftarrow \infty$
    for all $\alpha \in S_j$ **do**
      if $\exists a_u = (u, u_+) : \ell(a_u) = \alpha$ **then**
        if $\exists a_v = (v, v_+) : \ell(a_v) = \alpha$ **then**
          $\text{LCDS}_j[u, v] \leftarrow \min\{\text{LCDS}_j[u, v],\ w(a_u) + \text{LCDS}_{j+1}[u_+, v_+]\}$
          $\text{LCDS}_j[v, u] \leftarrow \min\{\text{LCDS}_j[v, u],\ w(a_u) + \text{LCDS}_{j+1}[v_+, u_+]\}$
        else
          $\text{LCDS}_j[u, v] \leftarrow \min\{\text{LCDS}_j[u, v], w(a_u) + w(u_+, t)\}$
        end if
      else
        if $\exists a_v = (v, v_+) : \ell(a_v) = \alpha$ **then**
          $\text{LCDS}_j[v, u] \leftarrow \min\{\text{LCDS}_j[v, u], w(a_v) + w(v_+, t)\}$
        end if
      end if
    end for
    if $w(r, v) \leq w(r, u)$ **then**
      if $w(r, u) + \min\{\text{LCDS}_j[u, v], \text{LCDS}_j[v, u]\} > z^* + \Delta$ **then**
        SoundReduce(j, u, v) ▷ First procedure below
        break
      end if
    else if $w(r, v) + \min\{\text{LCDS}_j[u, v], \text{LCDS}_j[v, u]\} > z^* + \Delta$ **then**
      SoundReduce(j, v, u) ▷ First procedure below
      break
    end if
  end for
end procedure

Subroutine: Sound-reduces node u into node v at level j

procedure SoundReduce(j, u, v)
  for all $a = (q, u) \in A$ **do**
    $a \leftarrow (q, v)$ ▷ Redirects arcs to v
  end for
  v.lhs $\leftarrow \emptyset$ ▷ Removes v's state
  RemoveIfDisconnected(j, u) ▷ Next procedure below
end procedure

Subroutine: Removes node $u \in U_j$ and subsequent disconnected nodes in D

procedure RemoveIfDisconnected(j, u)
  if $\nexists a = (v, u) \in A$ **then**
    $U_j \leftarrow U_j \setminus \{u\}$
    for all $a = (u, v) \in A$ **do**
      $A \leftarrow A \setminus \{a\}$
      RemoveIfDisconnected($j + 1, v$)
    end for
  end if
end procedure

9 Postoptimality Analysis

Because a sound decision diagram transparently represents all near-optimal solutions, a wide variety of postoptimality analyses can be conducted with minimal computational effort. We describe a few of these here.

Algorithm 6 Retrieves δ-optimal solutions from a sound diagram for $\delta \in [0, \Delta]$

```
1: function RetrieveSolutions(δ)
2:     Suf^δ(t) = {null}                          ▷ Suf^δ(u) = set of possible δ-suffixes of u
3:     w(null) = 0                                ▷ null is the zero-length suffix.
4:     for j = n → 1 do                           ▷ Retrieve δ-optimal solutions in bottom-up pass.
5:         for all u ∈ U_j do
6:             Suf^δ(u) = ∅
7:             for all a = (u, v) ∈ A_j do
8:                 for all s ∈ Suf^δ(v) do        ▷ Examine suffixes of v.
9:                     if w(r, u) + w(a) + w(s) ≤ z* + δ then   ▷ Possible new δ-suffix of u?
10:                        Suf^δ(u) ← Suf^δ(u) ∪ {ℓ(a)||s}      ▷ Append ℓ(a) to suffix s.
11:                        w(ℓ(a)||s) = w(ℓ(a)) + w(s)          ▷ Compute weight of new suffix.
12:                    end if
13:                end for
14:            end for
15:        end for
16:    end for
17:    return Suf^δ(r)                            ▷ Returns set of δ-optimal solutions.
18: end function
```

The most basic postoptimality task is to retrieve all feasible solutions whose cost is within a given distance of the optimal cost. That is, we wish to retrieve all δ-optimal solutions from a diagram D that is sound for $\mathrm{P}(\Delta)$, for a desired $\delta \in [0, \Delta]$. This is accomplished by Algorithm 6. The algorithm assumes that the weight $w(r, u)$ of a minimum-weight path from r to each node u has been pre-computed in a single top-down pass. Then, for each desired tolerance δ, the algorithm finds δ-optimal solutions in a bottom-up pass. It accumulates for each node u a set $\mathrm{Suf}^\delta(u)$ of suffixes that could be part of a δ-optimal solution, based on the weight of a minimum-weight r–u path. When the algorithm reaches the root r, $\mathrm{Suf}^\delta(r)$ is precisely the set $\mathrm{Suf}_\delta(r)$ of δ-optimal solutions, because at this point the exact cost of solutions is known.

The worst-case complexity of the algorithm is proportional to the number of solutions D represents, including spurious solutions. However, many spurious solutions are screened out as the algorithm works its way up, particularly because a solution with cost greater than $z^* + \delta$ (where possibly $\delta \ll \Delta$) can be discarded. Retrieval can therefore be quite fast for small δ.

The same algorithm can answer a number of postoptimality questions. For example, one might ask which solutions are δ-optimal when certain variables are fixed to certain values—or, more generally, when the domains S_j of certain variables are replaced by proper subsets S'_j of those domains. This is easily addressed by removing, for each S'_j, all arcs leaving layer j with labels that do not belong to S'_j. Algorithm 6 is then applied to the smaller diagram that results, after recomputing weights $w(r, u)$. Since the use of smaller domains does not add any Δ-optimal solutions, no δ-optimal solutions are missed. Methods for efficient updating of shortest paths are discussed in [35].

Another possible use of sound diagrams is to reoptimize when the objective function coefficients are altered, in which case the range of near-optimal

Algorithm 7 Computes δ-optimal domains for all variables, where $\delta \in [0, \Delta]$

```
1: procedure ComputeNearOptimalDomains(δ)
2:     for j = 1 → n do
3:         X_j ← ∅                          ▷ X_j is the subset of S_j in δ-optimal solutions.
4:         for all a = (u, v) ∈ A_j : ℓ(a) ∉ X_j do        ▷ Loops on arcs of missing values
5:             if w(r, u) + w(a) + w(v, t) ≤ x* + δ then
6:                 X_j ← X_j ∪ {ℓ(a)}       ▷ Found a δ-optimal solution where x_j = ℓ(a)
7:             end if
8:         end for
9:     end for
10: end procedure
```

solutions has to be restricted accordingly. If the cost coefficients are changed from c to c', then the objective function value of each solution may vary by at most $\gamma = \sum_{j=1}^{n} |c'_j - c_j|$, and thus in worst case we can assume that spurious solutions in the diagram have values above $z^* + \Delta - \gamma$, while the optimal value of the perturbed problem is at most $z^* + \gamma$. Therefore, the diagram remains sound for $\delta \leq \Delta - 2\gamma$ (for $\gamma \leq \frac{\Delta}{2}$). In a sense, we could regard the computational cost of constructing sound diagrams for larger values of Δ as a form of achieving faster re-optimization for a broader range of objective function coefficients.

Several additional types of postoptimality analysis can be performed, all with the advantage that spurious solutions have no effect on the computations. These types of analysis can therefore be conducted very rapidly.

For example, we can determine the values that a given variable can take such that the resulting minimum cost is within δ of the optimum. We refer to this as the *δ-optimal domain* of the variable. For each variable x_j, we need only scan the arcs leaving layer j and observe which ones pass test (2) when Δ is replaced with δ. This is done in Algorithm 7, which computes δ-optimal domains for all variables. In particular, the solution value of x_j is invariant across all δ-optimal solutions if its δ-optimal domain is a singleton. The algorithm assumes that shortest path lengths $w(r, u)$ and $w(u, t)$ have been pre-computed for each node u. Its complexity is dominated by the complexity $\mathcal{O}(nW^2)$ of computing the shortest path lengths. The algorithm can also be run after the domains of certain variables are replaced with proper subsets of those domains, to determine the effect on the δ-optimal domains of the other variables.

We can also perform range analysis for individual cost coefficients c_j. As with the previous analysis, the presence of spurious solutions has no effect. For each variable x_j, we can look for the values of c_j that would make each value in the domain of x_j optimal, provided that the other cost coefficients are unchanged. This idea is particularly simple and insightful when the domains are binary, as described in Algorithm 8: if there are solutions in the diagram for which $x_j = 0$ and $x_j = 1$, there is a unique value c'_j for which there are alternate optima with both values. Any $c_j > c'_j$ makes solutions where $x_j = 1$ suboptimal, and conversely any $c_j < c'_j$ makes solutions where $x_j = 0$

Algorithm 8 Computes the cost coefficient c'_j for each variable x_j on 0–1 domains that yields optimal solutions with $x_j = 0$ and $x_j = 1$ among the solutions of the decision diagram, if the other cost coefficients remain the same

```
1: function ComputeIndifferentCostCoefficients()
2:     for j = 1 → n do
3:         for α ∈ {0, 1} do
4:             z_α ← ∞                          ▷ z_α is min Σ_{j'≠j} c_{j'} x_{j'} when x_j = α
5:         end for
6:         for all a = (u, v) ∈ A_j do
7:             if w(r, u) + w(v, t) < z_{ℓ(a)} then
8:                 z_{ℓ(a)} ← w(r, u) + w(v, t)     ▷ Found a lower value of Σ_{j'≠j} c_{j'} x_{j'}
9:             end if
10:        end for
11:        if z_0 = ∞ then                      ▷ There is no solution with x_j = 0
12:            c'_j = ∞
13:        else if z_1 = ∞ then                 ▷ There is no solution with x_j = 1
14:            c'_j = −∞
15:        else                                 ▷ There are solutions for both assignments
16:            c'_j = z_0 − z_1                 ▷ Coefficient for which z_0 = z_1 + c'_j
17:        end if
18:    end for
19:    return c'
20: end function
```

suboptimal. If applied to solutions that were originally Δ-optimal, the outcome remains valid as long as c_j does not change more than Δ.

10 Computational Experiments

The experiments were designed to assess the compactness of sound-reduced decision diagrams, as well as the computational cost of constructing them. For demonstration purposes we implemented the above algorithms for 0–1 programming, which accounts for the great majority of pure integer problem instances in MIPLIB.[1]

The tests are based on 39 problem instances from MIPLIB 2.0, 3.0, and 2010. The instances were chosen to obtain a variety of problem sizes and characteristics, and they range in size from 9 to 10,757 variables. We constructed three data structures for each instance using one or more tolerances Δ. The first structure is a branching tree T that represents all Δ-optimal solutions. The second is the sound diagram U that is obtained from Algorithms 1–4, but omitting the sound-reduction step in Algorithm 5. The third is the sound-reduced diagram S that is obtained by applying Algorithms 1–5. Diagram S is therefore a smallest possible sound diagram for the problem instance. By comparing the size U with the size of T, we can see the advantage of

[1] For example, all but one of the 22 pure integer instances in MIPLIB 3.0, and all but 25 of the 146 pure integer instances in MIPLIB 2010, are 0–1 instances.

representing solutions with a sound decision diagram in which equivalent states are unified. By comparing the size of S with the size of U, we can measure the additional advantage obtained by sound reduction.

We carried out the experiments using a tolerance Δ that is quite large, so as to test the limits of the sound reduction procedure. We refer to this tolerance hereafter as $\Delta_{\max}$, which is adjusted to suit the problem instance as described below. For some instances, we constructed multiple diagrams for tolerances Δ ranging from 0 to $\Delta_{\max}$ in increments of $0.1\Delta_{\max}$, so as to investigate how the diagram sizes and computation times vary over a wide range of Δs.

We emphasize that the Δ used in practice to construct a diagram is likely to be considerably smaller than $\Delta_{\max}$, and the computation time correspondingly less. For instances larger than those tested here, the tolerance Δ can be reduced further to keep the computation time within practical bounds. This is likely to be necessary, in any case, to prevent the space of near-optimal solutions from growing too huge to be analyzed in any comprehensible way. We also note that a diagram need only be constructed once to answer a large number of postoptimality queries.

For the smaller instances, we set $\Delta_{\max}$ large enough to encompass all feasible solutions; that is, large enough so that all feasible solutions are $\Delta_{\max}$-optimal. For the remaining instances, we initially set $\Delta_{\max}$ equal to the median absolute value of nonzero objective coefficients. This allows us to test variations of up to 100% in objective coefficients of half of these variables. If the runtime was less that 1000 seconds, we kept doubling $\Delta_{\max}$ until the runtime exceeded 1000 seconds, but stopped short of a doubling that resulted in a runtime of more than 24 hours. Some additional problem instances are not reported here because either approach took too long to generate even the set of optimal solutions.

For most instances, we constructed the sound-reduced diagram S in two ways. (a) We first used the algorithm given above to generate all near-optimal solutions for the given Δ while building the diagram S . We refer to this as the *stand-alone method*, which we also used to construct U. (b) We then used CPLEX to generate all near-optimal solution and fed these solutions into a modification of the above algorithm to obtain S. The modified algorithm omits the mechanism that searches for Δ-optimal solutions. We refer to this as the *CPLEX-assisted method*. The results for the these instances using both methods appear in Table 2. For the remaining instances, which are generally harder to solve, we applied only the CPLEX-assisted method, and the results appear in Table 3.

The advantage of the stand-alone method is that the user need not have access to a particular commercial solver, and need only obtain the optimal value from any available solver. The advantage of the CPLEX-assisted method is that CPLEX tends to generate near-optimal solutions much more rapidly than our algorithm, because it can exploit the cutting planes and other information gathered during the initial solution of the problem.

In constructing both S and U, the branching priority is Depth-First Search (DFS), which consists of exploring the branching node with more fixed values

first. Variables are ordered by increasing index, and 0-arcs are explored before 1-arcs. The code is written in C++ (gcc version 4.8.24), ran in Ubuntu 14.04.2 LTS on a machine with Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz processors and 128 GB of RAM.

Table 2 shows, for each instance, the number of variables, the tolerance $\Delta_{\max}$, the number of optimal solutions, and the number of $\Delta_{\max}$-optimal solutions. Subsequent columns show the size of T, U, and S, followed by the construction time for U and S using the standalone method. Following this are the time requirements of the CPLEX-assisted method: the time required by CPLEX to generate the near-optimal solutions (labeled "CPLEX"), and the time required to construct a sound-reduced diagram that represents them (labeled "SR").

Finally, the table displays the time necessary to reoptimize after perturbing the cost coefficients: first by allowing CPLEX to re-solve the problem, and then by recomputing a shortest path in the sound-reduced diagram. We perturb the objective function coefficients by adding $\frac{\Delta}{10n}$ to each, where n is the number of variables. This ensures that optimal solutions for the new objective function correspond to Δ-optimal solutions for the original objective function.

Table 3 contains similar information, but without computation times for the stand-alone method and reoptimization. Computation times throughout are reported with two significant digits, since further precision is meaningless.

Tables 2 and 3 show that sound reduction yields a diagram S that is more compact than the branching tree T in every instance. The reduction in size is almost always substantial, and sometimes several orders of magnitude. In many cases, S contains even fewer nodes than the number of Δ-optimal solutions. S is also generally smaller than U, although one must normally pay a higher computational price for this reduction.

Figure 8 plots the number of nodes in T and S for all instances tested, ordered by increasing size of T. Since the vertical scale is logarithmic, the plot clearly indicates the substantial compression that is obtained for most instances. In addition, compression tends to be greater for harder instances, and the maximum size of S levels off at about a million nodes as the size of T grows.

Since the desired tolerance Δ is typically much less than $\Delta_{\max}$ in practice, we display in Figs. 9–10 how the diagram sizes and computation times depend on Δ for six of the instances. The diagram sizes and runtimes are plotted on a logarithmic scale. As predicted by Corollary 1, the diagram size is monotone nondecreasing in Δ. The figures indicate that one can substantially reduce the size and computational cost of a sound-reduced diagram by using a somewhat smaller Δ; that is, by focusing on solutions that are somewhat closer to the optimum. The sound-reduced diagrams for typical values of Δ are well within a practical size range for rapid postoptimality processing.

With respect to computation times, a striking pattern emerges. For most of the instances (23 out of 39), the time required to build a sound-reduced diagram in the CPLEX-assisted method is less than the time CPLEX requires to generate the near-optimal solutions, often orders of magnitude less. In only

three instances is it an order of magnitude greater. This implies that diagram construction tends to comprise an even smaller fraction of the time consumed by the stand-alone method, which requires longer to generate solutions. One can therefore obtain the advantages of postoptimality analysis with decision diagrams by paying a relatively small computational cost beyond that required to generate near-optimal solutions— even (in most cases) when they are generated with state-of-the-art commercial software.

11 Conclusion

We explored sound decision diagrams as a data structure for concisely and transparently representing near-optimal solutions of integer programming problems. We showed that for a given variable ordering, repeated application of a simple sound-reduction step yields a smallest possible sound diagram for any given discrete optimization problem. Based on this result, we stated an algorithm for constructing sound-reduced diagrams for integer programming problems. Given the optimal value of the problem obtained from any solver, the algorithm generates all near-optimal solutions and represents them in a sound-reduced diagram.

We discussed how sound-reduced diagrams permit several types of postoptimality analysis with modest computational effort. Spurious solutions in the diagram can easily be recognized and discarded during the analysis when necessary, but even this is frequently unnecessary.

Computational testing indicates that sound-reduced diagrams generally offer dramatic reductions in the space required to represent near-optimal solutions, relative to that required by a branching tree. For the MIPLIB instances tested, the resulting diagrams are well within a size range that permits rapid postoptimality processing. In particular, they enable re-optimization with new cost coefficients more rapidly than with a commercial solver. A variation of the sound reduction algorithm accepts near-optimal solutions from an external solver, which can compute them much more rapidly due to its access to information gathered during the initial solution of the problem. Even in this case, diagram construction usually requires less time than solution generation. This shows that postoptimality analysis based on a sound-reduced diagram normally incurs a relatively small computational cost beyond that of generating the near-optimal solutions, although there are occasional exceptions.

Sound decision diagrams can also be used for postoptimality analysis of problems with multiple objectives. One objective function can be used to construct the diagram, while other objective functions are evaluated by solving shortest path problems with the corresponding weights. If the near-optimal threshold used for diagram construction is large enough, the near-optimal solutions relative to the additional objectives are guaranteed to be represented in the diagram.

This study is inspired by the idea that solution of an optimization problem should be viewed more broadly than merely generating one or more optimal

solutions. Rather, it should be seen as transforming an opaque data structure that defines the problem but does not reveal its solutions, to a transparent data structure that provides ready access to optimal and suboptimal solutions of interest. We attempted to lay a foundation for this type of solution for pure integer programming. The results suggest that postoptimality analysis based on sound-reduced diagrams can extract significantly more information from a model than was previously feasible.

We have focused on pure integer programming, which is worthy of attention because it comprises an important subclass of problems. For example, 22 of 65 instances (34%) in MIPLIB 3.0 are pure integer, and 146 of 361 instances (40%) in MIPLIB 2010 are pure integer. In addition, a strong sound-reduction theorem can be proved for the pure integer case. Nonetheless it would be highly desirable to extend postoptimality analysis with decision diagrams to mixed integer programming. This remains a topic for future research.

Table 2: Computational results for 26 MIPLIB instances, showing construction times for the stand-alone and CPLEX-assisted methods.

			Solutions		Size (nodes)			Construction time (s)				Reoptimization (s)	
Instance	Variables	$\Delta_{\max}$	Opt.	$\Delta_{\max}$-opt.	T	U	S	U	S	CPLEX	SR	IP	SP
air01	771	5,400	2	117,997	33,060,654	142,554	142,554	1,200	36,000	1,300	5,000	0.26	0.041
air02	6,774	1,000	1	593	2,711,251	347,128	347,128	44,000	69,000	260	2,100	0.99	0.19
air03	10,757	1,000	1	56	335,521	106,765	106,765	20,000	11,000	240	9,200	1.9	0.1
air06	8,627	5	1	3	25,752	17,272	14,760	22,000	16,000	16	14,000	1.4	0.0057
bm23*	27	60	1	2,168	23,620	20,356	5,460	30	22	7.9	2.2	0.22	0.0012
enigma*	100	1	2	4	278	243	243	26	20	2.6	0.0071	1.0	0.000043
l152lav	1,989	30	1	37,741	16,558,175	326,111	223,882	3,200	39,000	1,400	4,500	2.4	0.18
lp4l	1,086	100	24	91,672	25,015,670	286,118	286,118	2,400	27,000	600	4,100	0.97	0.22
lseu	89	240	2	74,845	2,269,072	313,996	55,864	3,500	6,400	990	250	0.78	0.077
misc01b*†	82	250	8	600	6,305	2,972	2,645	2.4	3.8	1.3	0.17	0.34	0.0015
misc02b*†	58	1,500	4	1,168	11,190	3,204	2,580	1.6	2.8	5.1	0.25	0.25	0.001
misc03b*†	159	1,600	24	4,320	125,319	63,495	55,780	110	132	8.4	47	0.44	0.05
misc07b*†	259	400	72	16,272	1,196,731	693,436	309,571	3,000	7,100	200	1,600	1.4	0.33
mod008	319	30	6	22,685	3,986,603	915,342	117,603	82,000	42,000	22,000	300	2.7	0.17
mod010	2,655	4	128	62,272	26,017,798	289,812	289,812	4,500	74,000	4,200	15,000	0.73	0.28
p0033*	33	2,200	9	10,746	55,251	847	449	6.8	35	3.4	0.4	0.27	0.00035
p0040*	40	7,200	1	519,216	2,736,899	2,950	831	2.7	480	186	22	0.033	0.00032
p0201	201	400	4	52,120	3,389,643	141,101	8,761	1,200	7,200	400	44	0.64	0.0062
pipex*	48	200	1	12,266	299,558	95,805	49,843	130	760	43	290	0.71	0.038
sentoy	60	270	1	67,820	1,502,480	1,413,109	84,422	4,400	3,500	1,000	790	15	0.000032
stein9*	9	4	54	172	460	137	80	0.03	1.8	0.005	0.036	0.17	0.000038
stein15*	15	6	315	2,809	8,721	2,158	816	0.46	1.1	0.15	0.82	0.5	0.00044
stein27*	27	9	2,106	367,525	1,450,702	338,916	25,444	150	530	120	220	1.8	0.05
stein45*	47	0	70	70	2,530	2,410	1,766	320	165	71	0.15	5.8	0.0016

*$\Delta_{\max}$ is set large enough to include all feasible solutions.

† These instances correspond to `misc01`–`misc07` in MIPLIB, in which a single continuous variable is used for the sole purpose of representing the objective function value. We adapted these instances by removing such variable.

Table 3: Computational results for 13 MIPLIB instances, showing construction times for the CPLEX-assisted method.

			Solutions		Size (nodes)		Construction time (s)		Reoptimization (s)	
Instance	Variables	$\Delta_{\max}$	Opt.	$\Delta_{\max}$-opt.	T	S	CPLEX	SR	IP	SP
acc-tight5	1,339	1,000,000	247	321	382,944	333,553	1,300	450	17	0.39
air04	8,904	10	8	748	6,417,114	1,020,710	4,100	7,600	49	2.6
air05	7,195	70	2	486	3,037,250	1,067,368	1,000	3,500	62	3.8
bnatt350	3,150	10	38	38	7,9903	65,410	170,000	38	3,100	0.15
cap6000	6,000	40	1	64	297,819	112,034	10,000	250	5.2	0.0064
eil33-2	4,516	200	1	359	1,355,329	758,933	840	1,000	70	0.83
mine-90-10	900	100,000	1	3,441	517,374	3,269	7,800	120	140	0.00078
mine-166-5	830	500,000	1	6,749	1,391,928	4,106	390	250	30	0.018
mitre	10,724	14	80	80	454,002	190,016	110	880	3.3	0.44
neos-1109824	1,520	3	3,072	102,912	45,691,703	348,153	53,000	13,000	22	0.68
neos-1616732	200	2	86	5366	585,822	122,177	84,000	320	150	0.19
opm2-z7-s2	2,023	10	1	216	41,369	22,173	51,000	72	52	0.001
p0282	282	40	1	200,909	20,977,973	283	24,000	800	6.3	0.00023
p0291	291	10	1	295,162	25,952,774	292	36,000	920	0.38	0.00053
p6000	6,000	50	1	116	523,781	166,684	22,000	490	4.0	0.0064

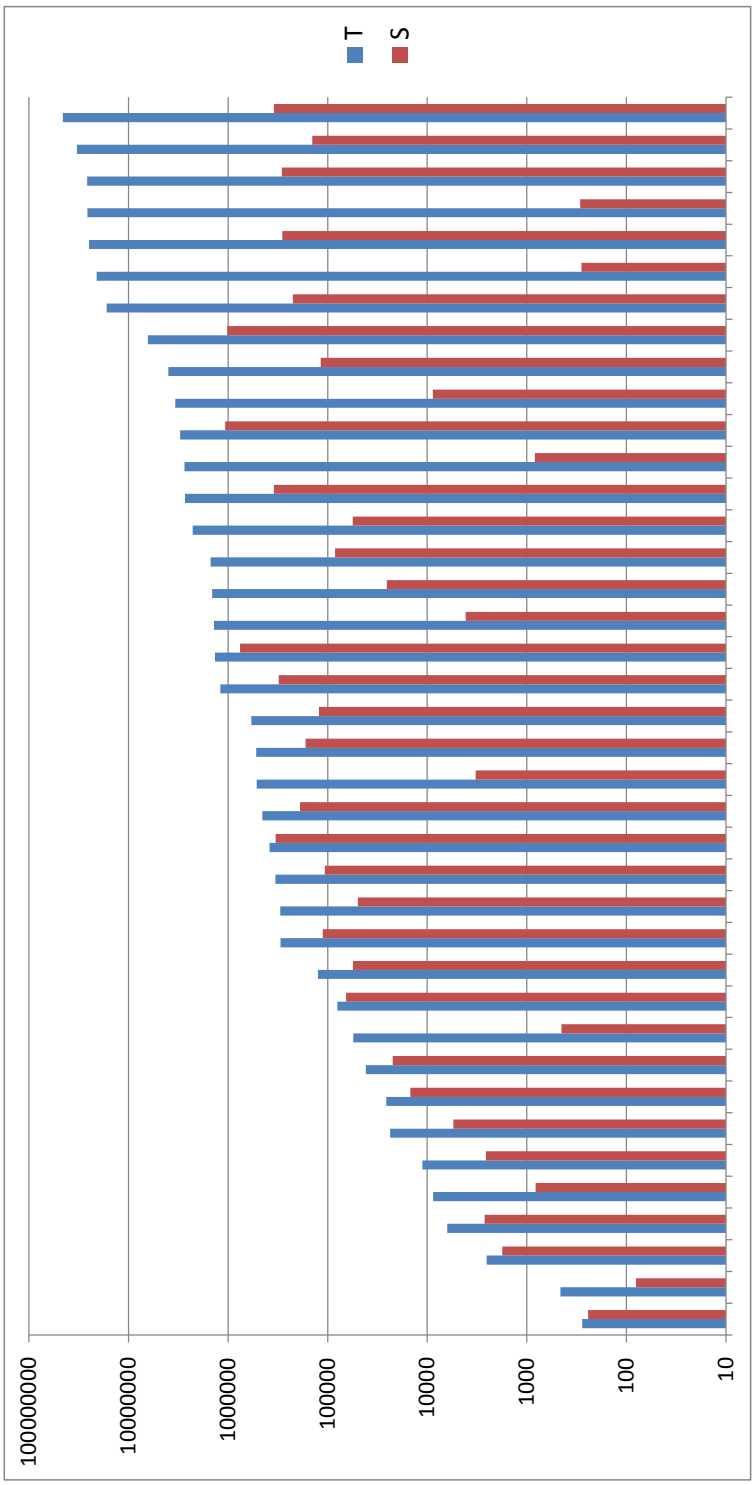

Fig. 8: Number of nodes in T and S for all instances tested, indicating the amount of compression.

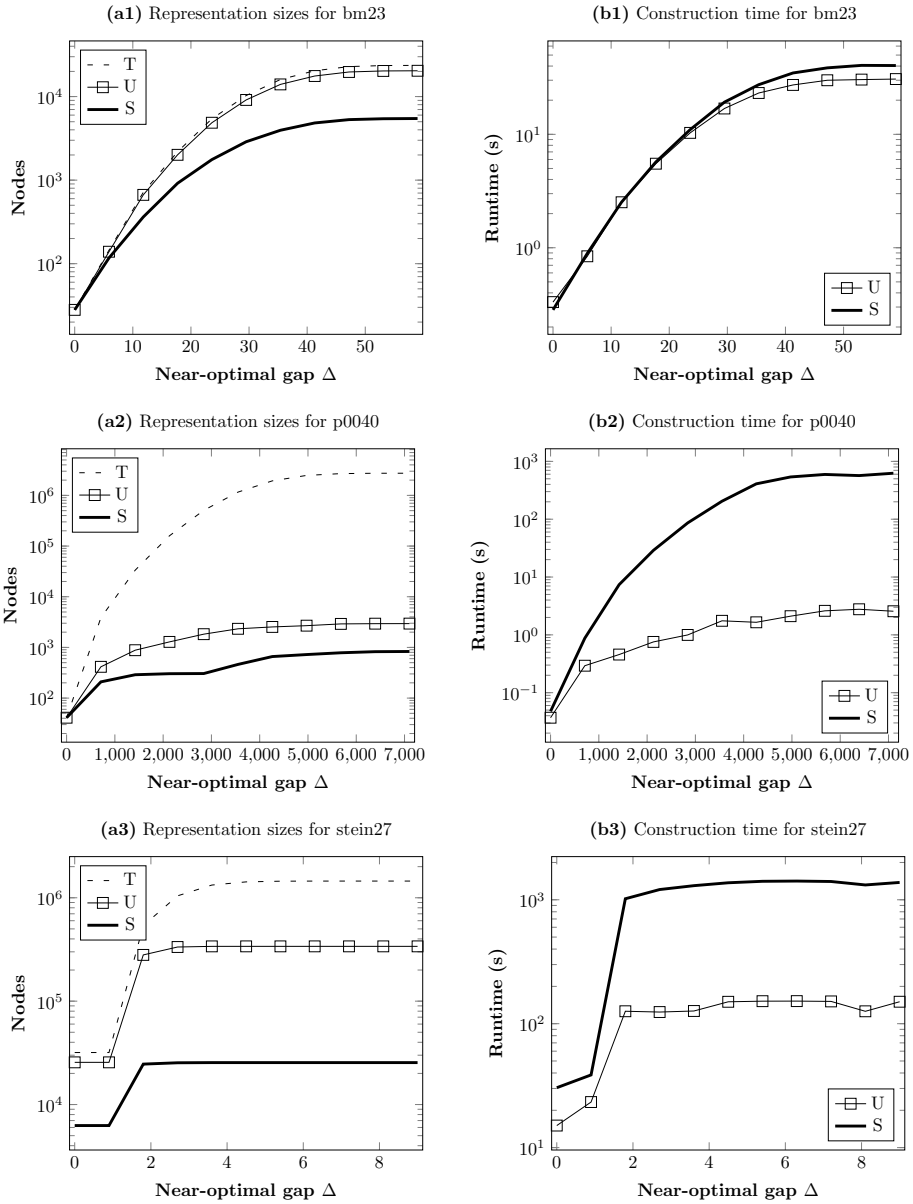

Fig. 9: Diagram size and computation time vs. Δ for three smaller instances.

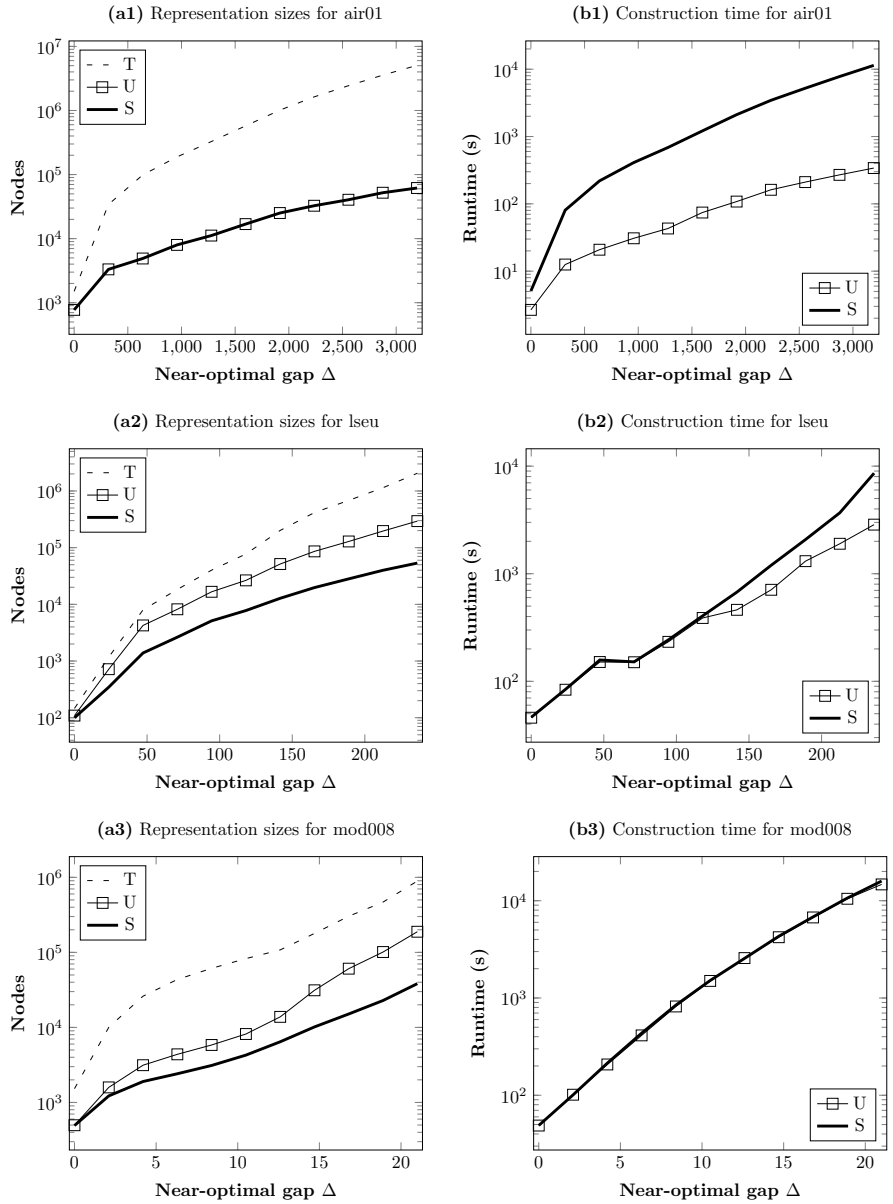

Fig. 10: Diagram size and computation time vs. Δ for three larger instances.

References

1. Achterberg, T., Heinz, S., Koch, T.: Counting solutions of integer programs using unrestricted subtree detection. In: L. Perron, M.A. Trick (eds.) Proceedings of CPAIOR, pp. 278–282. Springer (2008)
2. Akers, S.B.: Binary decision diagrams. IEEE Transactions on Computers **C-27**, 509–516 (1978)
3. Andersen, H.R., Hadžić, T., Hooker, J.N., Tiedemann, P.: A constraint store based on multivalued decision diagrams. In: C. Bessiere (ed.) Principles and Practice of Constraint Programming (CP 2007), *Lecture Notes in Computer Science*, vol. 4741, pp. 118–132. Springer (2007)
4. Arthur, J.A., Hachey, M., Sahr, K., Huso, M., Kiester, A.R.: Finding all optimal solutions to the reserce site selection problem: Formulation and computational analysis. Environmental and Ecological Statistics **4**, 153–165 (1997)
5. Behle, M.: Binary decision diagrams and integer programming. Ph.D. thesis, Universität des Saarlandes (2007)
6. Bergman, D., Cire, A.A., van Hoeve, W.J., Hooker, J.N.: Variable ordering for the application of bdds to the maximum independent set problem. In: CPAIOR (2012)
7. Bergman, D., Ciré, A.A., van Hoeve, W.J., Hooker, J.N.: Discrete optimization with binary decision diagrams. INFORMS Journal on Computing **28**, 47–66 (2016)
8. Bollig, B., Wegener, I.: Improving the variable ordering of OBDDs is NP-complete. IEEE Transactions on Computers **45**(9), 993–1002 (1996)
9. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **C-35**, 677–691 (1986)
10. Camm, J.D.: ASP, the art and science of practice: A (very) short course in suboptimization. Interfaces **44**(4), 428–431 (2014)
11. Danna, E., Fenelon, M., Gu, Z., Wunderling, R.: Generating multiple solutions for mixed integer programming problems. In: M. Fischetti, D.P. Williamson (eds.) Proceedings of IPCO, pp. 280–294. Springer (2007)
12. Dawande, M., Hooker, J.N.: Inference-based sensitivity analysis for mixed integer/linear programming. Operations Research **48**, 623–634 (2000)
13. Ebendt, R., Gunther, W., Drechsler, R.: An improved branch and bound algorithm for exact bdd minimization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **22**(12), 1657–1663 (2003)
14. Fischetti, M., Salvagnin, D.: Pruning moves. INFORMS Journal on Computing **22**(1), 108–119 (2010)
15. Fischetti, M., Toth, P.: A new dominance procedure for combinatorial optimization problems. Operations Research Letters **7**(4), 181–187 (1988)
16. Gamrath, G., Hiller, B., Witzig, J.: Reoptimization techniques for MIP solvers. In: E. Bampis (ed.) Proceedings of SEA, pp. 181–192. Springer (2015)
17. GAMS Support Wiki: Getting a list of best integer solutions of my MIP (2013). URL https://support.gams.com
18. Geoffrion, A.M., Nauss, R.: Parametric and postoptimality analysis in integer linear programming. Management Science **23**(5), 453–466 (1977)
19. Glover, F., Løkketangen, A., Woodruff, D.L.: An annotated bibliography for post-solution analysis in mixed integer programming and combinatorial optimization. In: M. Laguna, J.L. González-Velarde (eds.) OR Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, pp. 299–317. Kluwer (2000)
20. Goetzendorff, A., Bichler, M., Shabalin, P., Day, R.W.: Compact bid languages and core pricing in large multi-item auctions. Management Science **61**(7), 1684–1703 (2015)
21. Greistorfer, P., Løkketangen, A., Voß, S., Woodruff, D.L.: Experiments concerning sequential versus simultaneous maximization of objective function and distance. Journal of Heuristics **14**, 613–625 (2008)
22. Hadžić, T., Hooker, J.N.: Discrete global optimization with binary decision diagrams. In: Workshop on Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geometry (GICOLAG). Vienna (2006)

23. Hadžić, T., Hooker, J.N.: Postoptimality analysis for integer programming using binary decision diagrams. Tech. rep., Carnegie Mellon University (2006)
24. Hadžić, T., Hooker, J.N.: Cost-bounded binary decision diagrams for 0–1 programming. In: P. van Hentemryck, L. Wolsey (eds.) CPAIOR Proceedings, *Lecture Notes in Computer Science*, vol. 4510, pp. 332–345. Springer (2007)
25. Haus, U.U., Michini, C.: Representations of all solutions of boolean programming problems. In: ISAIM (2014)
26. Hoda, S., van Hoeve, W.J., Hooker, J.N.: A systematic approach to MDD-based constraint programming. In: Proceedings of the 16th International Conference on Principles and Practices of Constraint Programming, Lecture Notes in Computer Science. Springer (2010)
27. Holm, S., Klein, D.: Three methods for postoptimal analysis in integer linear programming. Mathematical Programming Study **21**, 97–109 (1984)
28. Hosaka, K., Takenaga, Y., Kaneda, T., Yajima, S.: Size of ordered binary decision diagrams representing threshold functions. Theoretical Computer Science **180**, 47–60 (1997)
29. Hu, A.J.: Techniques for efficient formal verification using binary decision diagrams. Thesis CS-TR-95-1561, Stanford University, Department of Computer Science (1996)
30. Ibaraki, T.: The power of dominance relations in branch-and-bound algorithms. Journal of the Association for Computing Machinery **24**(2), 264–279 (1977)
31. IBM Support: Using CPLEX to examine alternate optimal solutions (2010). URL http://www-01.ibm.com/support/docview.wss?uid=swg21399929
32. Kilinç-Karzan, F., Toriello, A., Ahmed, S., Nemhauser, G., Savelsbergh, M.: Approximating the stability region for binary mixed-integer programs. Operations Research Letters **37**, 250–254 (2009)
33. Kohler, W., Steiglitz, K.: Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. Journal of the Association for Computing Machinery **21**(1), 140–156 (1974)
34. Lee, C.Y.: Representation of switching circuits by binary-decision programs. Bell Systems Technical Journal **38**, 985–999 (1959)
35. Miller-Hooks, E., Yang, B.: Updating paths in time-varying networks given arc weight changes. Transportation Science **39**, 451–464 (2005)
36. Schrage, L., Wolsey, L.: Sensitivity analysis for branch and bound integer programming. Operations Research **33**, 1008–1023 (1985)
37. Van Hoesel, S., Wagelmans, A.: On the complexity of postoptimality analysis of 0/1 programs. Discrete Applied Mathematics **91**, 251–263 (1999)
38. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. Society for Industrial and Applied Mathematics (2000)