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

Compact Representation of Near-Optimal Integer Programming Solutions

Thiago Serra · J. N. Hooker

May 2017

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, once the optimal value is known. The structure of a decision diagram facilitates rapid processing of a wide range of queries about the near-optimal solution space. To obtain a more compact diagram, we exploit the property that such diagrams may become paradoxically smaller when they contain more solutions. We use sound decision diagrams, which innocuously admit some solutions that are worse than near-optimal. We describe a simple "sound reduction" operation that, when applied repeatedly in any order, yields a smallest possible sound diagram for a given problem instance. We find that sound reduction yields a structure that is typically far smaller than a tree that represents the same set of near-optimal solutions.

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

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

For example, in a recent study [7] 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 [14], 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. One 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 [2,6,21,24,28], but they can also compactly represent solutions of a discrete optimization problem [3,5,16,19]. 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.

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 2. The branching tree of Fig. 1(a) represents the three feasible solutions that have a value within 4 of the optimum, 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

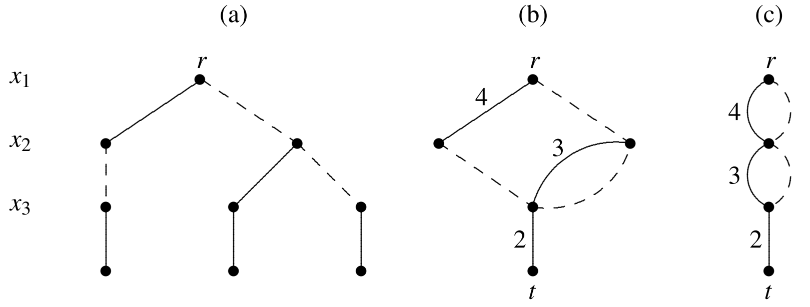

Fig. 1: (a) Branching tree for near-optimal solutions of (1). (b) Reduced weighted decision diagram representing the same solutions. (c) Sound decision diagram for (1).

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 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 [6].

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 [18], which is a diagram that represents all near-optimal solutions along with some *spurious* solutions whose objective function values are worse than near-optimal. The spurious solutions may be feasible or infeasible. By judiciously admitting spurious solutions into the diagram, one can significantly reduce its size while maintaining soundness of the near-optimal solution set.

In particular, we show that a certain *sound reduction* operation, which replaces a pair of nodes with a single node, yields a smaller sound diagram. Our main theoretical result is that repeated application of sound reduction operations, in any order, 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 call such a diagram *sound reduced.* A problem may have multiple sound-reduced diagrams, but they all have the same minimum size.

Sound diagrams have several advantages for postoptimality analysis. Aside from their smaller size, they allow for easy extraction of near-optimal solutions. One need only to enumerate paths in the diagram while discarding those that represent spurious solutions, which are easily identified by the fact that their values are too far from the optimum. In order to infer that solutions are spurious while constructing and when querying such diagrams, the optimal value is first obtained by solving the problem with a conventional solver.

For example, Fig. 1(c) illustrates a sound diagram for problem (1). It represents the three solutions within 4 of the optimal value, plus a spurious solution $(x_1, x_2, x_3) = (1, 1, 1)$ that is discarded because its value is greater than 6. This solution happens to be infeasible, but it is not necessary to check feasibility, which is time-consuming. It is only necessary to compute the path length.

A further advantage of sound diagrams is that the presence of spurious solutions has no effect whatever on the implementation or complexity of many types of postoptimality analysis. It is enough that the diagram represent all near-optimal solutions.

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. 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 [13] 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 [15], 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.

The "one-tree" method of [8] 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. While possible, the collection is not intended to be exhaustive, and there is no indication of how to represent the collection compactly or query more easily. A "branch-and-count" method is presented 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 as part of our mechanism for constructing sound decision diagrams. However, we focus on compact representation of near-optimal solutions.

The commercial solver CPLEX has offered a "solution pool" feature since version 11.0 [22] that relies on the one-tree method. The solution pool has been supported by the the GAMS modeling system since version 22.6 [11]. By contrast, postoptimality software based on decision diagrams operates apart from the solution method, requiring only the optimal value from the solver. It also differs by generating an exhaustive set of near-optimal solutions and organizing them in a decision diagram that is convenient for postoptimality analysis.

Integer programming sensitivity analysis has been investigated for some time, as for example in [9,10,12,20,23,26,27]. 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 [17], and the concept of a sound diagram was introduced in [18]. 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 $\text{Sol}(D)$ denote the set of solutions represented by the r–t paths. We say that D *exactly represents* S when $\text{Sol}(D) = S$.

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.

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 [6].

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

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'
$\mathrm{Sol}(D)$	set of solutions represented by r–t paths in diagram D
z^*	optimal value of problem (P)
$\mathrm{P}(\Delta)$	problem of finding Δ-optimal solutions of (P)
$S(\Delta)$	set of Δ-optimal solutions of (P)
$\mathrm{ILP}(\Delta)$	problem of finding Δ-optimal solutions of (ILP)
$\mathrm{Pre}(u)$	set of prefixes of node u
$\mathrm{Suf}(u)$	set of suffixes of node u
$\mathrm{Suf}_\Delta(u)$	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
lhs.u	left-hand-side state at node u
$\mathrm{LCDS}_j[u, v]$	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

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). Let x be a Δ-*optimal* solution of (P) when $x \in S$ 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 $\mathrm{P}(\Delta)$ denote the the problem of finding Δ-optimal solutions of (P).

We say that D exactly represents $\mathrm{P}(\Delta)$ when $\mathrm{Sol}(D) = S(\Delta)$. Since such a decision diagram can be quite large, we wish to identify smaller diagrams that approximately represent $\mathrm{P}(\Delta)$. We therefore study decision diagrams that are *sound* for $\mathrm{P}(\Delta)$, which represent a superset of $S(\Delta)$. Specifically, 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. 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 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, nodes and/or arcs 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 $S(\Delta)$.

It is easy to check whether 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

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

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. 2). 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, $\pi(u^*)$ and π' are both optimal r–u^* paths, and thus 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(Δ), then D is sound for P(δ) for any $\delta \in [0, \Delta]$.*

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

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

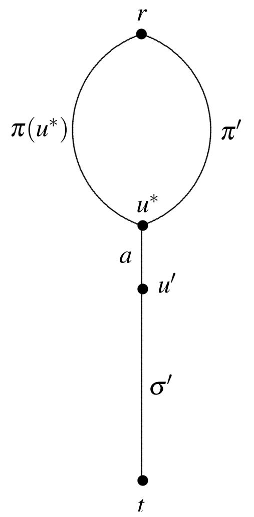

Fig. 2: Illustration of the proof of Theorem 1.

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:

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

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

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 3 illustrates sound reduction. Figure 3(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 3(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. 3(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* $\mathrm{Suf}_\Delta(u) \neq \mathrm{Suf}_\Delta(v)$.

Proof Suppose to the contrary that $\mathrm{Suf}_\Delta(u) = \mathrm{Suf}_\Delta(v)$, 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 $x(\pi) \in \mathrm{Pre}(u)$ and $x(\sigma) \in \mathrm{Suf}(v) \setminus \mathrm{Suf}(u)$, then $x(\sigma) \notin \mathrm{Suf}_\Delta(u)$, and therefore $x(\sigma) \notin \mathrm{Suf}_\Delta(v)$. 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. $\square$

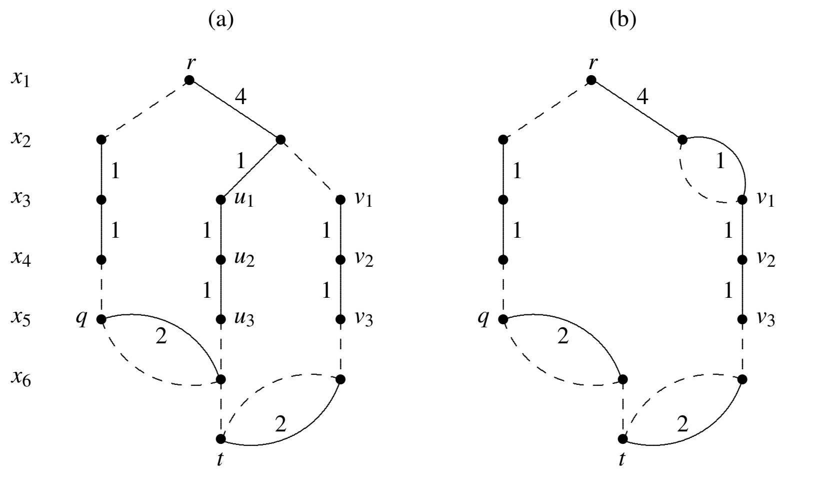

Fig. 3: Reduced (a) and sound-reduced (b) decision diagrams for $z^* = 2$ and $\Delta = 6$.

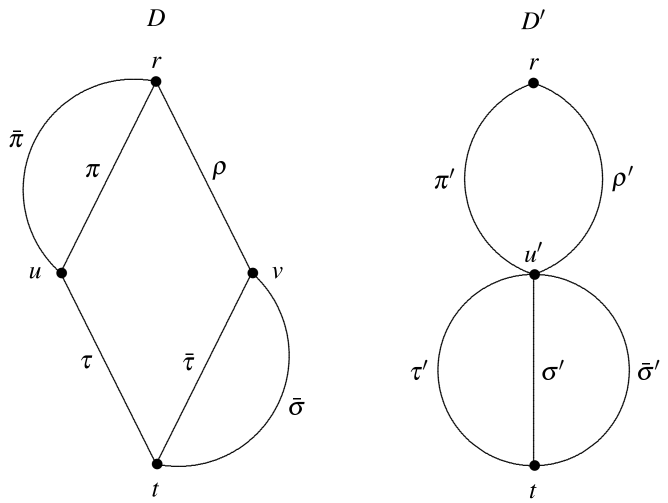

Fig. 4: Illustration of the proof of Lemma 4.

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* $\mathrm{Suf}_\Delta(u) = \mathrm{Suf}_\Delta(u')$.

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 $\mathrm{P}(\Delta)$, 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 $\mathrm{Suf}_\Delta(u) \subseteq \mathrm{Suf}_\Delta(u')$, due to the fact that π is a minimum-weight path. However, by hypothesis $\mathrm{Suf}_\Delta(u) \neq \mathrm{Suf}_\Delta(u')$, and so we have $\mathrm{Suf}_\Delta(u') \setminus \mathrm{Suf}_\Delta(u) \neq \emptyset$.

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 $x(\bar{\sigma}) \notin \mathrm{Suf}(u)$, 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 $\mathrm{P}(\Delta)$. Thus any sound diagram for $\mathrm{P}(\Delta)$ 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 $\mathrm{P}(\Delta)$, the node u' in layer j of D' with $\mathrm{Suf}_\Delta(u') = \mathrm{Suf}_\Delta(u)$ 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 $\mathrm{P}(\Delta)$ have minimum size, they are not necessarily identical. For example, while all sequences of sound reductions of Fig. 3(a) terminate in the same diagram Fig. 3(b), this is not the case for the slightly different diagram of Fig. 5(a). Sound-reducing u_3 into

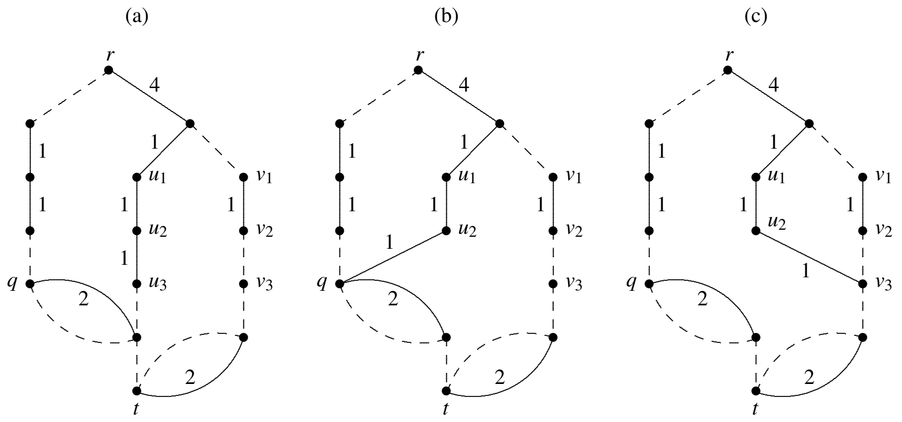

Fig. 5: Distinct sound-reduced diagrams (b) and (c) obtained from diagram (a), where $z^* = 2$ and $\varDelta = 6$.

q yields the sound-reduced diagram of Fig. 5(b), and sound-reducing u_3 into v_3 yields Fig. 5(c). Both diagrams are of minimum size and satisfy Lemma 4, but they are distinct.

6 Two-Sided Soundness

A sound diagram is permitted to represent feasible and infeasible solutions that are worse than $\varDelta$-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 $\varDelta$-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 $[z^*, z^* + \varDelta]$, we will say that it has *two-sided soundness*, meaning that it satisfies

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

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. Namely, find 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. Let $U_1 = \{r\}$, $U_j = \{u_{j1}, \ldots, u_{jn}\}$ for $j = 2, \ldots, n$, and $U_{n+1} = \{t\}$. There is one arc with weight s_k from r to each u_{2k}, $k = 1, \ldots, n$. There are two arcs from each u_{jk} to $u_{j+1,k}$ for $j = 2, \ldots, n-1$, one with weight zero, and the other with weight s_j if $j-1 < k$ 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-1} if $k = n$ and weight s_n otherwise. Then there is a one-to-one correspondence between r–t paths and nonempty subsets of S. 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 $x_j, \ldots, x_n$, 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 $\min\{u.\mathrm{lhs},\ b - M_j\}$ and still identify nodes that have the same state. Here, $M_j = (M_{1j}, \ldots, M_{mj})$, 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 $w(r,u) + \mathrm{LP}_j(u.\mathrm{lhs}) > z^* + \Delta$, where

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

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 $\mathrm{LP}_j(u.\mathrm{lhs})$ can underestimate the weight of u–t paths.

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 $(q.\mathrm{lhs}, w(r,u) + c_j \ell(a))$, where

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

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 $v.\mathrm{lhs} = q.\mathrm{lhs}$, and we let arc a run from u 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.`closed` = *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.`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.`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.`openNodes` reaches zero.

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

```
procedure PopulateSoundDiagram()
    r ← new Node(min{0, M_1}, 0)
    U_1 ← {r}
    PriorityQueue ← {(1, r)}                          ▷ Begins search at root node r
    RevisitBFSQueue ← {}
    while PriorityQueue ≠ ∅ do
        (j, u) ← PriorityQueue.pop()                  ▷ Queue policy defines search type
        u.openArcs ← |S_j|
        Deadend ← true
        for α ∈ S_j do
            Success ← TryBranching(j, u, α)           ▷ Algorithm 2
            Deadend ← Deadend ∧ ¬Success
        end for
        if Deadend then                               ▷ No branch succeeded
            RemoveDeadendNode(j, u)                   ▷ Procedure below
        else if RevisitBFSQueue ≠ ∅ then              ▷ Nodes following u reopened
            RevisitNodes()                            ▷ Algorithm 3
        else if u.openArcs = 0 then                   ▷ Nodes following u are all closed
            CloseNode(j, u)                           ▷ Algorithm 4
        end if
        u.explored ← true
    end while
end procedure
```

Subroutine: Removes nodes that cannot reach t recursively

```
procedure RemoveDeadendNode(j, u)
    U_j ← U_j \ {u}
    for all a = (v, u) ∈ A do
        A ← A \ {a}
        v.openArcs ← v.openArcs − 1
        if ∄a′ = (v, u′) ∈ A : u′ ≠ u then            ▷ Node above is a deadend
            RemoveDeadendNode(j − 1, v)
        else if v.openArcs = 0 then
            CloseNode(j − 1, v)
        end if
    end for
end procedure
```

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.

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

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

```
function TryBranching(j, u, α)
    nodeLhs ← min{u.lhs + A_j α, M_j}
    nodeWeight ← w(r, u) + c_j α
    if nodeWeight + LP_j(u.lhs) > z* + Δ then                ▷ LP = ∞ if infeasible
        return false
    end if
    if ∃v ∈ U_{j+1} : v.lhs = nodeLhs then                    ▷ Found node with same lhs
        if nodeWeight < w(r, v) then                ▷ Improves minimum cost path to r
            w(r, v) ← nodeWeight
            if v.explored then
                RevisitBFSQueue.add(j + 1, v)                   ▷ For Algorithm 3
            end if
        end if
        A ← A ∪ {(u, v)}
        if v.closed then                      ▷ Fails if not improving for a closed node
            u.openArcs ← u.openArcs − 1
        end if
    else                                                   ▷ Creates node for new lhs
        v ← new Node(nodeLhs, nodeWeight)
        U_{j+1} ← U_{j+1} ∪ {v}
        A ← A ∪ {(u, v)}
        if j < n then                             ▷ Adds non-terminal node to queue
            PriorityQueue.add(j + 1, v)
        else                                          ▷ First reached terminal node t
            v.closed ← true
            w(v, t) ← 0
        end if
    end if
    return true
end function
```

path from v to the terminal node. Sound reduction is therefore attempted only 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 $\mathrm{LCDS}_j[v,u]$. The computation of $\mathrm{LCDS}_j[v,u]$ and $\mathrm{LCDS}_j[u,v]$ 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 $w(r,u) + \mathrm{LCDS}_j[u,v] \leq z^* + \Delta$. Condition (4) is 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

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
```

$w(r,u) + \mathrm{LCDS}_j[v,u] \leq z^* + \varDelta$. We can therefore sound-reduce u into v when

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

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 $O(nW^2S_{\max})$ operations before node removals. Each call of the bottom procedure requires time $O(S_{\max})$, for a total time of $O(nWS_{\max})$.

Algorithm 4 Closes nodes and performs bottom-up processing recursively

```
procedure CloseNode(j, u)
    u.closed ← true
    for all a = (u, v) ∈ A do                          ▷ Computes minimum cost path to t
        w(u, t) ← min{w(u, t), c_j ℓ(a) + w(v, t)}
    end for
    if w(r, u) + w(u, t) > z* + Δ then                 ▷ Node is not minimal
        RemoveDeadendNode(i, u)                        ▷ Subroutine in Algorithm 1
    end if
    for all a = (u, v) ∈ A do
        if w(r, u) + c_j ℓ(a) + w(u, t) > z* + Δ then  ▷ Arc is not minimal
            A ← A \ {a}
        end if
    end for
    ClosingQueue ← {}
    for all a = (v, u) ∈ A do
        if ¬v.closed then
            v.openArcs ← v.openArcs − 1
            if v.openArcs = 0 then                     ▷ Closes node above
                ClosingQueue ← ClosingQueue ∪ {(j, u)}
            end if
        end if
    end for
    CompressDiagram(j, u)                              ▷ Algorithm ?? or 5
    while ClosingQueue ≠ ∅ do
        (j, u) ← ClosingQueue.pop()
        CloseNode(j, u)
    end while
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.

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

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

```
procedure CompressDiagram(j, u)
    for all v ∈ U_j : v ≠ u ∧ v.closed, and v ordered by nondecreasing w(r, d) do
        LCDS_j[u, v], LCDS_j[v, u] ← ∞
        for all α ∈ S_j do
            if ∃a_u = (u, u_+) : ℓ(a_u) = α then
                if ∃a_v = (v, v_+) : ℓ(a_v) = α then
                    LCDS_j[u, v] ← min{LCDS_j[u, v], w(a_u) + LCDS_{j+1}[u_+, v_+]}
                    LCDS_j[v, u] ← min{LCDS_j[v, u], w(a_u) + LCDS_{j+1}[v_+, u_+]}
                else
                    LCDS_j[u, v] ← min{LCDS_j[u, v], w(a_u) + w(u_+, t)}
                end if
            else
                if ∃a_v = (v, v_+) : ℓ(a_v) = α then
                    LCDS_j[v, u] ← min{LCDS_j[v, u], w(a_v) + w(v_+, t)}
                end if
            end if
        end for
        if w(r, v) ≤ w(r, u) then
            if w(r, u) + min{LCDS_j[u, v], LCDS_j[v, u]} > z* + Δ then
                SoundReduce(j, u, v)                         ▷ First procedure below
                break
            end if
        else if w(r, v) + min{LCDS_j[u, v], LCDS_j[v, u]} > z* + Δ 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) ∈ A do
        a ← (q, v)                                           ▷ Redirects arcs to v
    end for
    v.lhs ← ∅                                                ▷ 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 ∄a = (v, u) ∈ A then
        U_j ← U_j \ {u}
        for all a = (u, v) ∈ A do
            A ← A \ {a}
            RemoveIfDisconnected(j + 1, v)
        end for
    end if
end procedure
```

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

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

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

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 [25].

One might also ask which solutions are δ-optimal when the objective function coefficients are altered, say to c'. Since changing the cost coefficients can introduce Δ-optimal solutions, the sound diagram may fail to represent some solutions that are Δ-optimal for the altered costs. However, we can identify all solutions that remain δ-optimal after the cost change for $\delta \leq \Delta - \sum_{j=1}^{n} |c'_j - c_j|$ since all of those were originally Δ-optimal. This is accomplished simply by modifying the arc weights to reflect the new costs and running Algorithm 6, again with recomputed weights $w(r, u)$.

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

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

procedure ComputeNearOptimalDomains(δ)
    for $j = 1 \to n$ **do**
        $X_j \leftarrow \emptyset$ ▷ X_j is the subset of S_j in δ-optimal solutions.
        for all $a = (u, v) \in A_j : \ell(a) \notin X_j$ **do** ▷ Loops on arcs of missing values
            if $w(r, u) + w(a) + w(v, t) \leq x^* + \delta$ **then**
                $X_j \leftarrow X_j \cup \{\ell(a)\}$ ▷ Found a δ-optimal solution where $x_j = \ell(a)$
            end if
        end for
    end for
end procedure

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

function ComputeIndifferentCostCoefficients()
    for $j = 1 \to n$ **do**
        for $\alpha \in \{0, 1\}$ **do**
            $z_\alpha \leftarrow \infty$ ▷ z_α is $\min \sum_{j' \neq j} c_{j'} x_{j'}$ when $x_j = \alpha$
        end for
        for all $a = (u, v) \in A_j$ **do**
            if $w(r, u) + w(v, t) < z_{\ell(a)}$ **then**
                $z_{\ell(a)} \leftarrow w(r, u) + w(v, t)$ ▷ Found a lower value of $\sum_{j' \neq j} c_{j'} x_{j'}$
            end if
        end for
        if $z_0 = \infty$ **then** ▷ There is no solution with $x_j = 0$
            $c'_j = \infty$
        else if $z_1 = \infty$ **then** ▷ There is no solution with $x_j = 1$
            $c'_j = -\infty$
        else ▷ There are solutions for both assignments
            $c'_j = z_0 - z_1$ ▷ Coefficient for which $z_0 = z_1 + c'_j$
        end if
    end for
    return c'
end function

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 $x_j = 0$ 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 decision diagrams, based on 0–1 problem instances in MIPLIB. We constructed three data structures for each of 12 instances and a range of 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 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 for tolerances Δ that range over a wide interval from zero to $\Delta_{\max}$ in increments of $0.1\Delta_{\max}$. 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. These instances are `bm23`, `enigma`, `p0033`, `p0040`, `stein9`, `stein15`, and `stein27`. Thus for these instances, $\Delta_{\max}$ is the difference in value between the best and worst solutions. For the remaining instances, we 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.

In all experiments, the branching priority is DFS, 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), uses the COIN-OR CLP solver[1] (version 1.16.10), and 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 displays the statistics for the maximum tolerance $\Delta_{\max}$, including the number of optimal and $\Delta_{\max}$-optimal solutions, and the size and construction time for T, U, and S. The sound-reduced diagram S is dramatically smaller than the branching tree T for all the instances except `enigma`. It is also significantly smaller than the unified diagram U except in the cases of `air01` and `enigma`, and smaller by at least an order of magnitude in three instances. On the other hand, sound reduction added significantly more computation time to the diagram construction in seven of the instances.

In practice, the desired tolerance Δ is typically much less than $\Delta_{\max}$. We therefore display in Figs. 6–7 how the diagram sizes and computation times depend on Δ for six of the instances. Note that the diagram sizes and runtimes are plotted on a logarithmic scale. As predicted by Corollary 1, the diagram size is monotone nondecreasing in Δ.

[1] projects.coin-or.org/Clp

Table 2: Solution counts, with diagram sizes and construction times, for MIPLIB instances using a maximum tolerance $\Delta_{\max}$.

Instance	$\Delta_{\max}$	Solutions		Size (nodes)			Runtime (s)	
		Opt.	$\Delta_{\max}$-opt.	T	U	S	U	S
air01	3194	2	16,899	5,058,113	61,652	61,652	340	11,000
bm23*	59	1	2,168	23,620	20,356	5,460	31	40
enigma*	1	2	4	278	243	243	41	41
lseu	236.16	2	67,250	2,057,264	294,108	53,465	2,900	8,600
mod008	21	6	4,954	891,543	188,359	38,292	15,000	16,000
p0033*	2112	9	10,746	55,251	847	449	5.8	33
p0040*	7102	1	519,216	2,736,899	2,950	831	2.6	620
p0201	375	4	34,504	2,326,052	107,312	6,627	1,900	7,700
sentoy	280.8	1	85,401	1,868,562	1,754,681	101,618	3,800	12,000
stein9*	4	54	172	460	137	80	0.02	0.05
stein15*	6	315	2,809	8,721	2,158	816	0.54	1.6
stein27*	9	2,106	367,525	1,450,702	338,916	25,444	159	1,400

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

The sound-reduced diagram S is substantially more compact than the branching tree T in every instance. It is also smaller than U in all instances but one, although one must normally pay a higher computational price for this reduction. Of course, a sound-reduced diagram need only be generated once in order to carry out a large number of postoptimality queries. The sound-reduced diagrams for typical values of Δ are well within a practical size range for rapid postoptimality processing, normally a few hundred or a few thousand nodes. The computation times for constructing diagrams of this size are likewise modest, ranging from a few seconds to a few minutes.

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 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. We showed how the resulting diagrams permit several types of postoptimality analysis, and that the presence of spurious solutions in the diagrams has no effect on most types of analysis. 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.

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

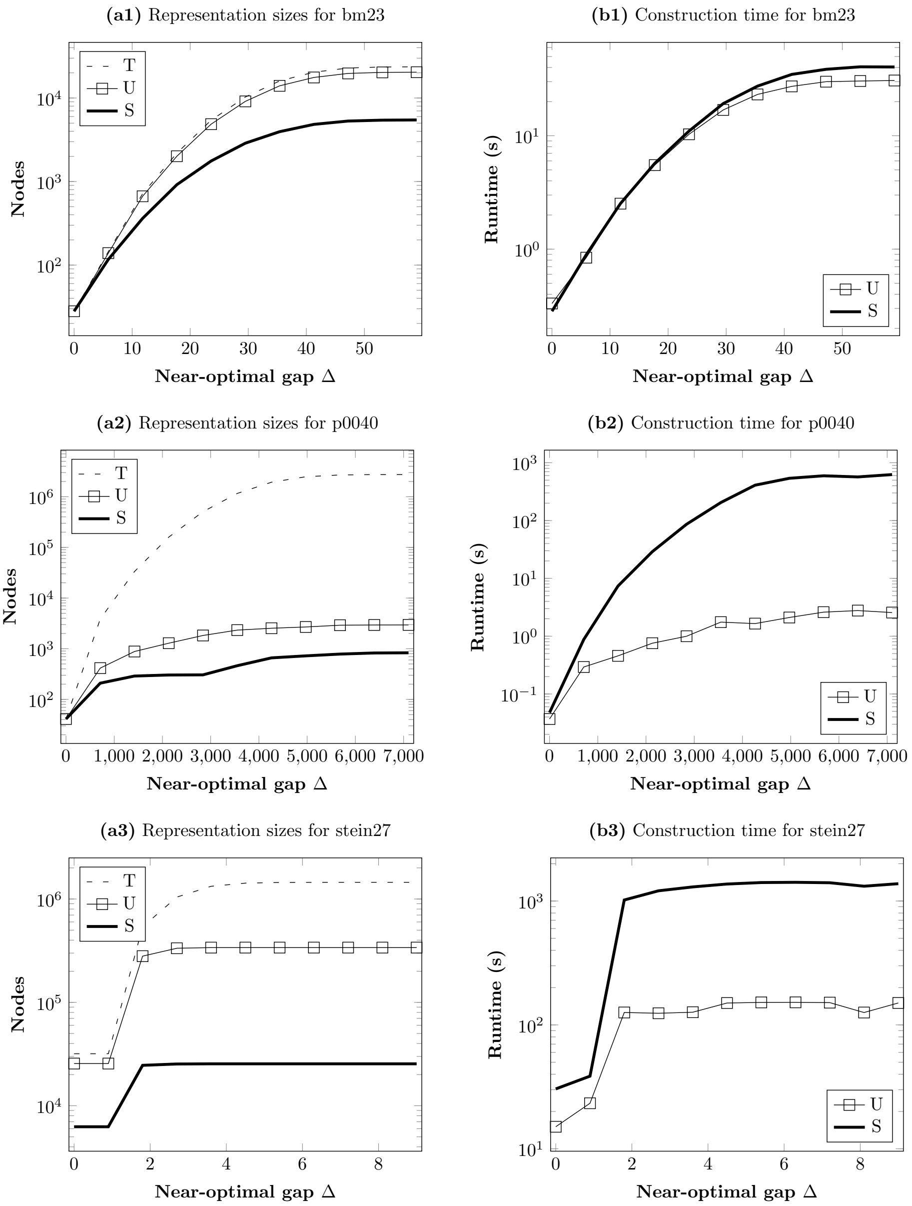

Fig. 6: Diagram size and computation time vs. Δ for three smaller instances.

interest. We attempted to lay a foundation for this type of solution for integer programming, but an obvious research direction is to extend the method to mixed integer programming. Decision diagrams can continue to play a role,

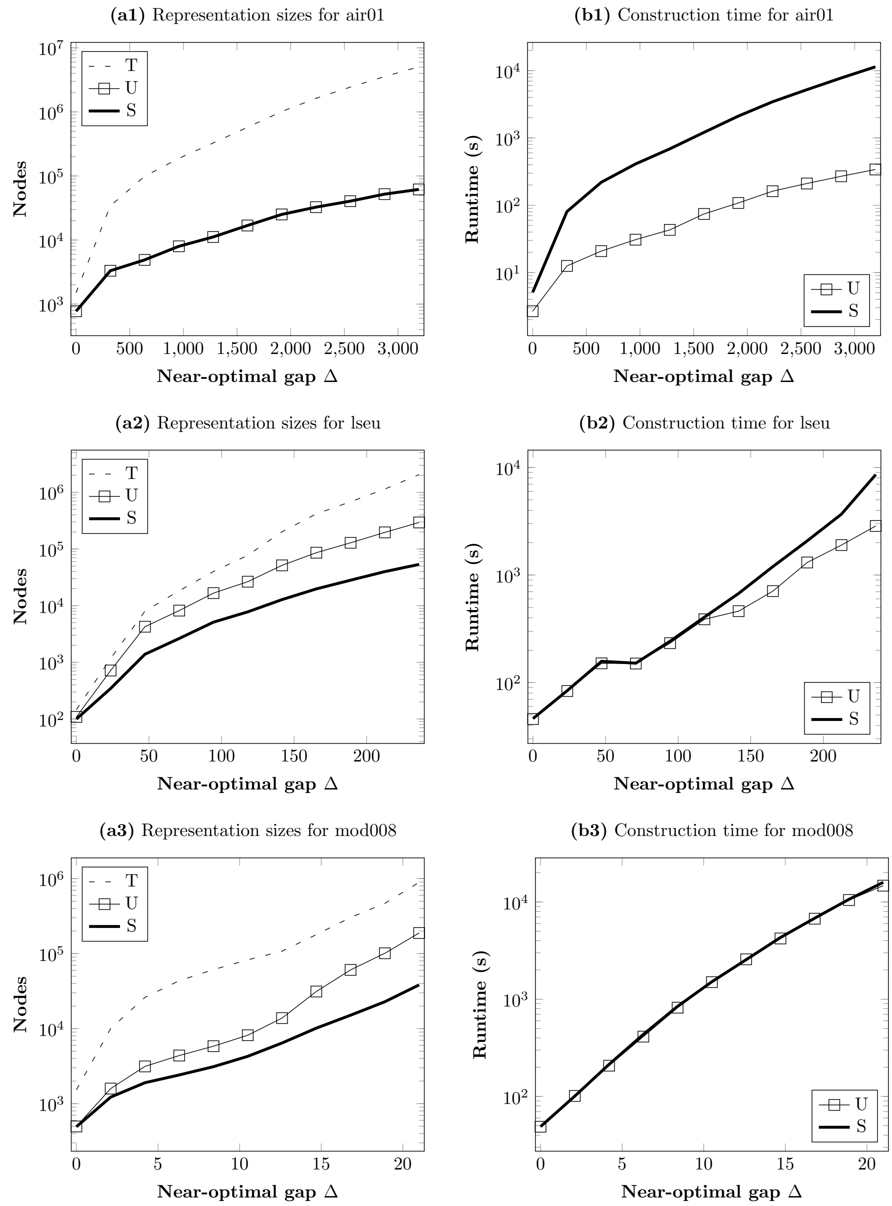

Fig. 7: Diagram size and computation time vs. Δ for three larger instances.

because paths in a diagram can represent values for the integer variables in the problem.

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. 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)
6. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **C-35**, 677–691 (1986)
7. Camm, J.D.: ASP, the art and science of practice: A (very) short course in suboptimization. Interfaces **44**(4), 428–431 (2014)
8. 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)
9. Dawande, M., Hooker, J.N.: Inference-based sensitivity analysis for mixed integer/linear programming. Operations Research **48**, 623–634 (2000)
10. Gamrath, G., Hiller, B., Witzig, J.: Reoptimization techniques for MIP solvers. In: E. Bampis (ed.) Proceedings of SEA, pp. 181–192. Springer (2015)
11. GAMS Support Wiki: Getting a list of best integer solutions of my MIP (2013). URL https://support.gams.com
12. Geoffrion, A.M., Nauss, R.: Parametric and postoptimality analysis in integer linear programming. Management Science **23**(5), 453–466 (1977)
13. 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)
14. 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)
15. 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)
16. 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)
17. Hadžić, T., Hooker, J.N.: Postoptimality analysis for integer programming using binary decision diagrams. Tech. rep., Carnegie Mellon University (2006)
18. 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)
19. 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)
20. Holm, S., Klein, D.: Three methods for postoptimal analysis in integer linear programming. Mathematical Programming Study **21**, 97–109 (1984)
21. Hu, A.J.: Techniques for efficient formal verification using binary decision diagrams. Thesis CS-TR-95-1561, Stanford University, Department of Computer Science (1996)

22. IBM Support: Using CPLEX to examine alternate optimal solutions (2010). URL http://www-01.ibm.com/support/docview.wss?uid=swg21399929
23. 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)
24. Lee, C.Y.: Representation of switching circuits by binary-decision programs. Bell Systems Technical Journal **38**, 985–999 (1959)
25. Miller-Hooks, E., Yang, B.: Updating paths in time-varying networks given arc weight changes. Transportation Science **39**, 451–464 (2005)
26. Schrage, L., Wolsey, L.: Sensitivity analysis for branch and bound integer programming. Operations Research **33**, 1008–1023 (1985)
27. Van Hoesel, S., Wagelmans, A.: On the complexity of postoptimality analysis of 0/1 programs. Discrete Applied Mathematics **91**, 251–263 (1999)
28. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. Society for Industrial and Applied Mathematics (2000)