Logic, Optimization, and Artificial Intelligence

J. N. Hooker
July 2025

Abstract Logic and optimization can, in combination, make valuable contributions to rule-based AI. Logic is the obvious medium for encoding a rule base and drawing inferences from it, while optimization provides a powerful technology for computing inferences. Their combination has taken on new relevance amid a growing concern for transparency in AI. Rule-based AI provides a natural solution to transparency that is becoming increasingly practical due to today's highly advanced optimization methods. This article surveys several areas of logic-optimization partnership, including probabilistic logic, Bayesian logic, belief logics and Dempster-Shafer theory, nonmonotonic (default) logic, many-valued logics, and inference of logical formulas from noisy data based on boolean regression. It shows how to compute projections, the fundamental problem of both logic and optimization, using decision diagrams and logic-based Benders decomposition. It concludes by describing how solvers can use postoptimality analysis to explain how conclusions are reached, further enhancing transparency.

Carnegie Mellon University, e-mail: jh38@andrew.cmu.edu

1 Introduction

Logic and optimization have played an essential role in artificial intelligence (AI) since its inception. The very first AI program (1956), *Logic Theorist* [74], proved theorems of logic and mathematics, including several from Russell and Whitehead's famous treatise *Principia Mathematica*. The following year, the pathbreaking AI system *General Problem Solver* [73] applied means-end analysis in a manner similar to logic programming. As for optimization, the advent of machine learning and artificial neural networks immediately posed the problem of adjusting parameters to fit training data. This optimization problem remains a central and challenging element of machine learning today.

Logic and optimization are not only individually useful in AI, but they have much to offer when combined. Together, they provide an ideal approach to implementing rule-based AI. While connectionist machine learning has achieved impressive results, it increasingly poses the issue of transparency, which is important for reproducibility, explainability, trustworthiness, and fairness [22, 44, 85, 89]. Rule-based AI provides a natural solution to transparency, because it allows knowledge to be explicitly encoded in rules from which relevant information can be inferred, and allows decisions to be deduced from a rule-based production system. While neural networks provide a useful model of cognition, a rule-based framework has long served as an alternative model. The well-known ACT-R framework inspired by AI pioneer Alan Newell, for example, places the agent's executive function in a rule-based production system [7, 63, 72].

Logic and optimization are natural partners in a rule-based environment. Logic provides the obvious framework for stating rules and deducing their implications, and even for their ethical assessment [60]. Optimization comes into play as a powerful method for computing logical inferences, as well as conceptual elucidation of the various logics used in AI. Optimization solvers, particularly for linear and integer programming, not only deduce inferences but can provide a traceback of how they are derived, thus enhancing the natural transparency of rule-based AI. These solvers have also improved enormously over the years. For example, one study documents that integer programming solvers reduced solution times by a factor of 5 million between 1989 and 2024, quite apart from reductions due to faster machines [84]. More fundamentally, we will see that optimization and logical inference are special cases of the same basic problem.

This paper presents a sampling of logic-optimization partnerships that have developed in parallel with machine learning and deserve renewed consideration, especially in view of today's advanced optimization technology and concern for transparency. We begin with probabilistic logic, which has a linear programming formulation that can be solved by modern column generation methods. We also describe various extensions, including Bayesian logic, which combines probabilistic logic with Bayesian networks and makes use of nonlinear programming. We examine belief logics as well, including Dempster-Shafer theory and variants that have easily-solved linear programming formulations. We then take up two nonstandard logics that have seen application in AI, default logic (a variety of nonmonotonic reasoning)

and many-valued logics. Both benefit from integer programming models. At this point we turn from deducing implications of logical formulas to inferring the formulas themselves from noisy data. We describe a form of boolean regression that can be calibrated by integer programming and measures statistical significance. We then delve into the underlying connection between logic and optimization just mentioned by observing that both, at root, pose a projection problem. We show how this problem can be addressed by applying a recently developed optimization method, logic-based Benders decomposition, to a knowledge base represented as a binary decision diagram (BDD). We conclude by specifying three ways in which optimization methods can provide tracebacks that are potentially useful for enhancing transparency in AI. One is classical linear programming sensitivity analysis, and the other two are inference-based and BDD-based postoptimality analysis for integer programming.

The emphasis throughout is on conveying the basic ideas and their motivation, rather than providing a complete technical description. Details are available in the cited references, which cover both the methods discussed here and subsequent developments.

2 Probabilistic Logic

Although George Boole is best known for his work in propositional logic, he regarded probabilistic logic [15] as his most important contribution. It in fact displays stunning originality. Although forgotten for more than a century, it is highly relevant to the project of artificial intelligence today. It allows one to determine with what confidence one can draw inferences from propositions that are known only to be true with certain probabilities.

Hailperin [45] observed in 1976 that Boole's probabilistic logic can be given a linear programming model. A decade after Hailperin's work, Nilsson independently published a similar model in the AI literature [75]. His paper gave rise to a number of subsequent contributions, many of which are surveyed in [24, 46, 62].

We begin by describing Boole's probabilistic logic and its linear programming model. We show how the well-known technique of column generation can deal with the exponentially many variables in the model. We then incorporate second-order probabilities without sacrificing linearity. Conditional independence assumptions introduce nonlinear constraints, but we indicate how exploiting the structure of a Bayesian network can somewhat ameliorate the resulting computational challenge.

2.1 The Basic Model

Probabilistic logic is best explained by an example [24]. Suppose the probabilities of three logical propositions are given as follows:

$$\Pr(x_1) = 0.9 \tag{1}$$
$$\Pr(x_1 \supset x_2) = 0.8 \tag{2}$$
$$\Pr(x_2 \supset x_3) = 0.7 \tag{3}$$

We are also given the conditional probability

$$\Pr(x_1 \mid x_2, x_3) = \frac{\Pr(x_1, x_2, x_3)}{\Pr(x_2, x_3)} = 0.8 \tag{4}$$

Note that $\Pr(x_1 \supset x_2)$ is not the conditional probability $\Pr(x_2 \mid x_1)$. It is the probability of the material conditional $x_1 \supset x_2$, which is equivalent to $\neg x_1 \vee x_2$. We wish to determine with what probability we can infer x_3. Boole observed that we cannot deduce a precise probability for x_3, but we can deduce that its probability lies in a particular *range*.

In probabilistic logic, each possible assignment of truth values to $\boldsymbol{x} = (x_1, x_2, x_3)$ is regarded as a "possible world" (values 0 and 1 correspond to false and true). Each possible world $\boldsymbol{x}$ has a probability $p_{\boldsymbol{x}}$, initially unknown. For example, p_{000} is the probability of the world in which $(x_1, x_2, x_3) = (0, 0, 0)$, and analogously for $p_{001}, \ldots, p_{111}$. Then the probability 0.9 of proposition x_1, for example, is

$$p_{100} + p_{101} + p_{110} + p_{111}$$

since x_1 is true in the corresponding possible worlds, and similarly for propositions $x_1 \supset x_2$ and $x_2 \supset x_3$. The probability assignments (1)–(3) can be written as the linear equations

$$p_{100} + p_{101} + p_{110} + p_{111} = 0.9 \tag{5}$$
$$p_{000} + p_{001} + p_{010} + p_{011} + p_{110} + p_{111} = 0.8 \tag{6}$$
$$p_{000} + p_{001} + p_{011} + p_{100} + p_{101} + p_{111} = 0.7 \tag{7}$$

The conditional probability assignment (4) can be linearized as

$$\Pr(x_1, x_2, x_3) = 0.8\Pr(x_2, x_3)$$

and therefore written

$$p_{111} - 0.8(p_{011} + p_{111}) = 0 \tag{8}$$

We can now deduce the range of possible values for the probability of x_3, which is equal to

$$p_{001} + p_{011} + p_{101} + p_{111} \tag{9}$$

A sharp upper bound on this probability is the maximum of (9) subject to the constraints (5)–(8) and the normalization and nonnegativity constraints

$$p_{000} + p_{001} + p_{010} + p_{011} + p_{100} + p_{101} + p_{110} + p_{111} = 1 \tag{10}$$
$$p_{000}, \ldots, p_{111} \geq 0 \tag{11}$$

This is a linear programming problem with maximum value 0.7. A sharp lower bound on the probability of x_3 is the minimum of (9) subject to these constraints, which is 0.5. The probability of x_3 must therefore be confined to the interval $[0.5, 0.7]$. Due to the convexity of the feasible set defined by the constraints, every probability in this interval is consistent with the assigned probabilities (5)–(8).

In general, the probabilistic inference problem is

$$\min/\max \ \left\{ \boldsymbol{c}^\top \boldsymbol{p} \;\middle|\; A\boldsymbol{p} = \boldsymbol{\pi},\ B\boldsymbol{p} = \mathbf{0},\ \mathbf{1}^\top \boldsymbol{p} = 1,\ \boldsymbol{p} \geq \mathbf{0} \right\} \tag{12}$$

where constraint $A\boldsymbol{p} = \boldsymbol{\pi}$ specifies the categorical probabilities, $B\boldsymbol{p} = \mathbf{0}$ specifies the conditional probabilities, and $\mathbf{1} = (1, 1, \ldots, 1)$. The probability assignments are unsatisfiable if (12) has no feasible solution. The example problem in this form is

$$\min/\max \left\{ [0\ 1\ 0\ 1\ 0\ 1\ 0\ 1]\boldsymbol{p} \;\middle|\; \begin{array}{l} \begin{bmatrix} 0&0&0&0&1&1&1&1 \\ 1&1&1&1&0&0&1&1 \\ 1&1&0&1&1&1&0&1 \end{bmatrix} \boldsymbol{p} = \begin{bmatrix} 0.9 \\ 0.8 \\ 0.7 \end{bmatrix} \\ [0\ 0\ 0\ -0.8\ 0\ 0\ 0\ 0.2]\boldsymbol{p} = 0 \\ [1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]\boldsymbol{p} = 1, \quad \boldsymbol{p} \geq \mathbf{0} \end{array} \right\} \tag{13}$$

where $\boldsymbol{p} = (p_{000}, \ldots, p_{111})$. If desired, probability ranges rather than point values can be specified in (12) while preserving the linear structure of the problem.

2.2 Column Generation

A difficulty with problem (12) is that it contains 2^n variables, where n is the number of atomic propositions x_i. One might think that this makes the problem computationally intractable, and Nilsson suggests as much in [76]. However, the solution technique *column generation*, well known to the optimization community, is designed for just such cases. It allows one to solve the problem by means of the simplex method while generating only a small fraction of the columns in matrices A and B, and thereby using only a small fraction of the variables in $\boldsymbol{p}$.

Column generation is readily applied to probabilistic logic, as initially observed in [20, 41, 48, 54, 59]. It begins with a restriction of (12) in which A and B contain only a few columns. It associates dual variables $\boldsymbol{u}$, $\boldsymbol{v}$, and α with the constraints $A\boldsymbol{p} = \boldsymbol{\pi}$, $B\boldsymbol{p} = \mathbf{0}$, and $\mathbf{1}^\top \boldsymbol{p} = 1$, respectively. These dual variables, whose values are calculated in the course of the simplex method, are used to compute the *reduced cost* of each variable p_j:

$$c_j - \boldsymbol{u}^\top A_j - \boldsymbol{v}^\top B_j - \alpha \tag{14}$$

where $(c_j, A_j, B_j, 1)$ is the column of $(\boldsymbol{c}^\top, A, B, \mathbf{1}^\top)$ corresponding to p_j. Supposing for the moment that we are solving the minimization problem, each iteration of the simplex method selects a new column $(c_j, A_j, B_j, 1)$ with a negative reduced cost for inclusion in the problem (when maximizing, a positive reduced cost is desired). If

no such column remains, an optimal solution has been found, at which point it is likely that only a small fraction of the columns in $(\boldsymbol{c}^\top, A, B, \mathbf{1}^\top)$ have been selected.

The problem of finding a p_j with a negative reduced cost, known as the *pricing problem*, can be solved by minimizing (14) over all columns j. The nature of the pricing problem is best illustrated in the example. A column of $(\boldsymbol{c}^\top, A, B, \mathbf{1}^\top)$ can be represented as $(y_0, y_1, y_2, y_3, z_1 - 0.8w_1, 1)$, where the 0–1 variables y_i, z_1 and w_1 are defined by

$$\begin{aligned} y_0 &\equiv x_3 \\ y_1 &\equiv x_1 \\ y_2 &\equiv (x_1 \supset x_2) \\ y_3 &\equiv (x_2 \supset x_3) \\ z_1 &\equiv (x_1 \wedge x_2 \wedge x_3) \\ w_1 &\equiv (x_1 \wedge x_2) \end{aligned} \tag{15}$$

The pricing problem can now be stated as minimizing

$$y_0 - (u_1 y_1 + u_2 y_2 + u_3 y_3) - v_1(z_1 - 0.8w_1) - \alpha \tag{16}$$

subject to (15), which is a special case of the maximum satisfiability problem. Solution of this problem yields a column with the smallest reduced cost, given by (16). In practice, the problem is first solved by a fast heuristic method. If this fails to yield a negative reduced cost, the problem is solved to optimality to confirm that no columns with a negative reduced cost remain. The simplex method terminates when this is confirmed.

One way to solve the pricing problem to optimality is to formulate it as a 0–1 programming problem as proposed in [48], which allows one to take advantage of powerful integer programming solvers. The formulas in (15) are converted to linear equations and inequalities, and (14) then optimized subject to these constraints. Conversion is straightforward. For example, $y_0 \equiv x_3$ becomes simply $y_0 = x_3$, and the remaining formulas are expressed in conjunctive normal form before conversion to inequalities. For instance, formula $y_2 \equiv (x_1 \supset x_2)$ becomes the conjunction of logical clauses

$$\neg y_2 \vee \neg x_1 \vee x_2, \quad y_2 \vee x_1, \quad y_2 \vee \neg x_2$$

which in turn become the three inequalities

$$(1 - y_2) + (1 - x_1) + x_2 \geq 1, \quad y_2 + x_1 \geq 1, \quad y_2 + (1 - x_2) \geq 1$$

Another approach is to formulate the pricing problem as pseudoboolean optimization, as proposed in [20, 54]. In the example, this is accomplished by replacing variables y_i, z_1, and w_2 in the objective function (15) with possibly nonlinear expressions in terms of the x_is as follows:

$$
\begin{aligned}
y_0 &\leftarrow x_3 \\
y_1 &\leftarrow x_1 \\
y_2 &\leftarrow 1 - x_1 x_2 \\
y_3 &\leftarrow 1 - x_2 x_3 \\
z_1 &\leftarrow x_1 x_2 x_3 \\
w_1 &\leftarrow x_2 x_3
\end{aligned}
$$

Then (15) is minimized without constraints, using a heuristic method, or when a proof of optimality is required, using a pseudoboolean optimization method such as those described in [18].

Subsequent developments in computational methods can be found in [47, 61, 62]. Some computationally easy cases of probabilistic logic are described in [4, 6].

2.3 Extensions

Problem (12) is easily modified to derive the range of a conditional probability. The desired conditional probability can be written $\boldsymbol{c}^\top \boldsymbol{p} / \boldsymbol{d}^\top \boldsymbol{p}$ for suitable coefficient vectors $\boldsymbol{c}$ and $\boldsymbol{d}$. This results in the linear-fractional programming problem

$$\min/\max \left\{ \frac{\boldsymbol{c}^\top \boldsymbol{p}}{\boldsymbol{d}^\top \boldsymbol{p}} \;\middle|\; A\boldsymbol{p} = \boldsymbol{\pi},\ B\boldsymbol{p} = \boldsymbol{0},\ \boldsymbol{1}^\top \boldsymbol{p} = 1,\ \boldsymbol{p} \geq \boldsymbol{0} \right\} \tag{17}$$

A simple change of variable, and the introduction of a scalar variable t, transforms (17) to a linear programming problem [25]. By setting $\boldsymbol{q} = t\boldsymbol{p}$, it is easily seen that (17) is equivalent to

$$\min/\max \left\{ \boldsymbol{c}^\top \boldsymbol{q} \;\middle|\; A\boldsymbol{q} = t\boldsymbol{\pi},\ B\boldsymbol{q} = \boldsymbol{0},\ \boldsymbol{1}^\top \boldsymbol{q} = t,\ \boldsymbol{d}^\top \boldsymbol{q} = 1,\ \boldsymbol{q} \geq \boldsymbol{0},\ t \geq 0 \right\} \tag{18}$$

When a solution $(\boldsymbol{q}, t)$ of (18) is found, the corresponding solution of (17) is $\boldsymbol{p} = \boldsymbol{q}/t$, with optimal value $\boldsymbol{c}^\top \boldsymbol{p} / \boldsymbol{d}^\top \boldsymbol{p} = \boldsymbol{c}^\top \boldsymbol{q}$. As an illustration, suppose we wish to find a probability range for $\Pr(x_3 | x_1, x_2)$ in the above example. In this case,

$$\frac{\boldsymbol{c}^\top \boldsymbol{p}}{\boldsymbol{d}^\top \boldsymbol{p}} = \frac{[0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]\boldsymbol{p}}{[0\ 0\ 0\ 0\ 0\ 0\ 1\ 1]\boldsymbol{p}}$$

The deduced interval for $Pr(x_3 | x_1, x_2) = \boldsymbol{c}^\top \boldsymbol{q}$ is a point value, namely 4/7.

A second extension arises when the given probabilities π_i are intervals rather than point values. Then the inference problem (12) is

$$\min/\max \left\{ \boldsymbol{c}^\top \boldsymbol{p} \;\middle|\; \boldsymbol{\pi}^{\text{lo}} \leq A\boldsymbol{p} \leq \boldsymbol{\pi}^{\text{hi}},\ B\boldsymbol{p} = \boldsymbol{0},\ \boldsymbol{1}^\top \boldsymbol{p} = 1,\ \boldsymbol{p} \geq \boldsymbol{0} \right\} \tag{19}$$

A probability range for $\boldsymbol{c}^\top \boldsymbol{p}$ can be derived by linear programming as before, but it might be regarded as too conservative. Probabilities at the ends of a range $[\pi_i^{\text{lo}}, \pi_i^{\text{hi}}]$ assigned to a premise are generally less likely to be the true probability than those near the middle. Probabilities at the ends of the derived range may therefore be

extremely unlikely, because they typically occur only when a number of the given probabilities are near the ends of their intervals. The derived range can therefore be misleadingly wide.

One possible remedy for this problem is to derive a "most likely" probability for $\boldsymbol{c}^\top \boldsymbol{p}$ by computing a maximum entropy solution [58]. Yet this poses a difficult nonlinear optimization problem [46], and it is equally misleading, because the true probability of the inferred proposition may be quite far from its most likely value. Another remedy, described in [24], is to specify a second-order probability distribution over each probability interval $[\pi_i^{\text{lo}}, \pi_i^{\text{hi}}]$ to indicate that probabilities near the middle of the interval are more likely. To preserve the linearity of the optimization problem, we can suppose the logarithm of the second-order probability density function is piecewise linear and therefore the lower envelope of a set of lines:

$$\log \Pr(\pi_i = \rho) = \min_{k=1,\ldots,k_i} \{\alpha_{ik}\rho + \beta_{ik}\}$$

Let A^i be row i of A, which corresponds to proposition i. Assuming independence of the second order probabilities, the logarithm of the joint probability that the m propositions have probabilities $A^1\boldsymbol{p}, \ldots, A^m\boldsymbol{p}$ is

$$\sum_{i=1}^{m} \min_{k=1,\ldots,k_i} \{\alpha_{ik} A^i \boldsymbol{p} + \beta_{ik}\} \tag{20}$$

To eliminate solutions that are extremely unlikely, one can impose a lower bound L on (20). The probabilistic inference problem becomes the linear programming problem

$$\min/\max \left\{ \boldsymbol{c}^\top \boldsymbol{p} \;\middle|\; \begin{array}{l} \sum_{i=1}^m \ell_i \geq L,\ \ell_i \leq \alpha_{ik} A^i \boldsymbol{p} + \beta_{ik},\ k = 1,\ldots,k_i,\ i = 1,\ldots,m \\ \boldsymbol{\pi}^{\text{lo}} \leq A\boldsymbol{p} \leq \boldsymbol{\pi}^{\text{hi}},\ B\boldsymbol{p} = \boldsymbol{0},\ \boldsymbol{1}^\top \boldsymbol{p} = 1,\ \boldsymbol{p} \geq \boldsymbol{0} \end{array} \right\}$$

which can again be solved by column generation. The probability range can now be reduced to a more realistic interval by increasing L. Second-order distributions of the conditional probabilities can be similarly accommodated. A similar model in [5] shows how to assign probabilities point values that are uncertain, where the degree of uncertainty is based on information from several sources.

2.4 Bayesian Logic

One of our most valuable sources of probabilistic knowledge is the independence of most events. It is convenient, for example, that the probability of an earthquake in Asia is independent of the weather in North America. Ironically, the independence relations that make things easier to understand also make a probabilistic logic model harder to solve, because they introduce nonlinearities. Despite this, it would be desirable to incorporate independence assumptions into Boole's probabilistic logic.

Baysian logic accomplishes this by identifying the nodes of a Bayesian network with logical formulas [3], thereby taking advantage of the elegant way in which Bayesian networks represent conditional independence.

Figure 1 represents a small Bayesian network for formulas $f_1, \ldots, f_6$. We can suppose that probabilities are specified (or bounded) for some or all of the nodes, conditioned on the node's immediate predecessors, as in the conditional probability $\Pr(f_1|f_2, f_3)$. Marginal probabilities can be specified (or bounded) for nodes without predecessors, such as f_5 and f_6. The formulas f_i contain propositional variables x_j as in probabilistic logic. The formulas are therefore related logically as well as through conditional probabilities.

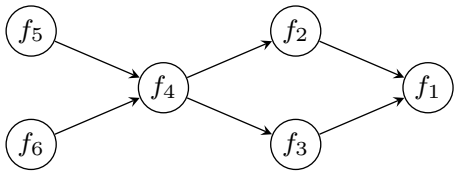

Fig. 1 A simple Bayesian network.

The essence of a Bayesian network is a Markovian property according to which conditional probabilities depend only on immediate parent nodes. Thus in Fig. 1, $\Pr(f_1|f_2, f_3, f_4) = \Pr(f_1|f_2, f_3)$. Consequently, a set of nodes with common parents are conditionally independent when their probabilities are conditioned on their parents. For example, f_2 and f_3 are conditionally independent given f_4, so that $\Pr(f_2|f_3, f_4) = \Pr(f_2|f_4)$. Since f_4 is a proposition, one may wish to suppose that f_2 and f_3 are independent given $\neg f_4$ as well as given f_4. We will indicate this by writing $\Pr(f_2|f_3F_4) = \Pr(f_2|F_4)$, where F_4 varies over f_4 and $\neg f_4$.

An optimization model is obtained by adding network-encoded nonlinear independence conditions to the linear programming model for probabilistic logic. For example, the independence assumption $\Pr(f_2|f_3F_4) = \Pr(f_2|F_4)$ is captured by the constraints

$$\Pr(f_2, f_3, F_4)\Pr(F_4) = \Pr(f_2, F_4)\Pr(f_3, F_4), \quad \text{for } F_4 = f_4, \neg f_4 \tag{21}$$

where $\Pr(f_2, f_3, F_4)$ is the joint probability of f_2, f_3, and F_4. Such constraints are not only nonlinear but can grow exponentially in number. If the probabilities are conditioned on several formulas, one must write constraints for all true-false combinations of these formulas.

Fortunately, it is shown in [3, 24] that the number of constraints is substantially limited in many networks. The idea is illustrated in Figs. 2 and 3. To compute bounds on the probability of the rightmost node, the number of independence constraints is at worst exponential in the size of the largest *extended ancestral set* of that node. Figure 2 shows the ancestral sets, and Fig. 3 the extended ancestral sets. The ancestral

sets are defined recursively in terms of parent sets. Given a set S of nodes, the parent sets of S are obtained by identifying the set S' of nodes outside S that are immediate predecessors of some node in S, and then splitting S' into maximal dependent subsets. The ancestral sets of a node are its parent sets, the parent sets of its parent sets and so forth. An extended ancestral set consists of an ancestral set augmented by the nodes in its parent sets.

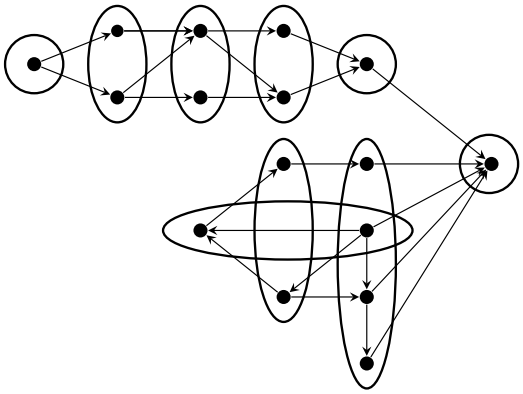

Fig. 2 Ancestral sets of the rightmost node.

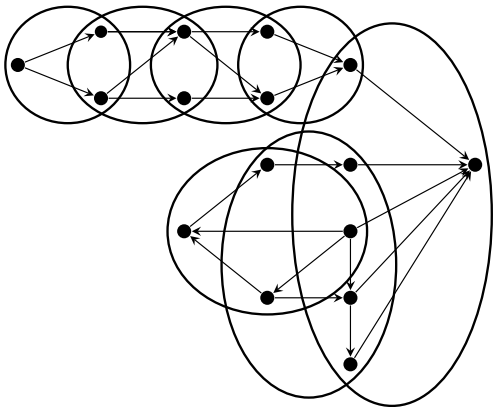

Fig. 3 Extended ancestral sets of the rightmost node.

A variation of Bayesian logic was recently proposed for a *logical credal network* [68]. Credal networks are modified Bayesian networks in which probabilities are specified in various imprecise ways, generalizing the intervallic specifications used here. A logical credal network associates logical formulas with nodes as does

Bayesian logic. In [68], the Markovian property of conventional Bayesian networks is replaced by a modified property that permits the network to contain cycles. Exact and heuristic solution methods for computing probability bounds in logical credal networks are described in [31, 67, 68]. Credal networks in general, including optimization methods for solving them, are comprehensively surveyed in [69].

3 Belief Logics

Like probabilistic logic, a logic of belief functions distributes probability mass over subsets of possible worlds. It differs in that the probability mass indicates the degree of evidence for, or credibility of, propositions rather than their probability in a classical sense. It might therefore be referred to as evidence mass. A further difference is that, whereas assigning a specific probability mass to a set of possible worlds *fixes* the total probability of those worlds, assigning a specific evidence mass *adds to* the total evidence for those worlds.

We begin below with a basic belief logic and its linear programming formulation. Unlike probabilistic logic, it does not contain exponentially many variables and therefore has no need of column generation for its solution. We then describe how Dempster-Shafer theory augments this basic model with a rule for combining possibly contradictory evidence sources (Dempster's combination rule). Following this, we drop the nonlinear independence assumption of Dempster-Shafer theory, while retaining Dempster's combination rule, to permit a linear programming formulation.

3.1 Logic of Belief Functions

Belief functions can be illustrated by a small example, taken from [3]. Suppose we have propositions x_1, x_2, and $x_1 \wedge x_2$, which correspond to possible world sets $S_1 = \{(1,0),(1,1)\}$, $S_2 = \{(0,1),(1,1)\}$, and $S_0 = \{(1,1)\}$, respectively. We have some evidence for x_1 and x_2 and wish to determine the implied evidential support for $x_1 \wedge x_2$. Let $m(S_1)$ be the fraction of available evidence that specifically supports the proposition x_1 corresponding to set S_1, and similarly for $m(S_2)$ and $m(S_0)$. We also suppose that $m(\Theta)$ is the fraction of evidence that supports no particular proposition, where Θ is the set of all possible worlds. Dempster and Shafer refer to $m(S_i)$ as a "basic probability number," but we will refer to it as an *evidence allotment*. We must have $\sum_i m(S_i) = 1$.

The accumulated evidence for x_1 consists of the evidence that supports x_1 and any propositions that imply x_1, in this case $x_1 \wedge x_2$. Thus the total support for set S_1 is $Bel(S_1) = m(S_0) + m(S_1)$, where Bel is a *belief function*. The belief values $Bel(S_1)$ and $Bel(S_0)$ are similarly defined. If we are given that $Bel(S_1) = 0.9$ and $Bel(S_2) = 0.8$, we have

$$\begin{aligned} &m(S_0) + m(S_1) = 0.9 \\ &m(S_0) + m(S_2) = 0.8 \\ &m(S_0) + m(S_1) + m(S_2) + m(\Theta) = 1 \\ &m(S_i) \geq 0, \text{ all } i \end{aligned} \tag{22}$$

The evidence allotments $m(S_i)$ must satisfy these constraints. Since evidence is allocated to no proper subset of S_0, the evidential support for $x_1 \wedge x_2$ is $Bel(S_0) = m(S_0)$. This support level must fall in the range obtained by maximizing and minimizing $m(S_0)$ subject to (22), which yields the interval $[0.7, 0.8]$.

In general, suppose we are given belief values $Bel(S_i)$ for $i = 1, \ldots, m$. The linear programming model is

$$\min/\max \left\{ \sum_{S \subseteq S_0} m(S) \;\middle|\; \begin{array}{l} \displaystyle\sum_{S \subseteq S_i} m(S) = Bel(S_i), \;\; i = 1, \ldots, m \\ \displaystyle m(\Theta) + \sum_{i=1}^{m} m(S_i) = 1 \\ m(\Theta) \geq 0, \;\; m(S_i) \geq 0, \;\; i = 1, \ldots, m \end{array} \right\}$$

Note that possible worlds do not play an explicit role in this model. The model therefore grows only linearly with the number of propositions assigned belief values and can therefore be solved very rapidly. In practice, set inclusion $S \subseteq S_i$ can be interpreted as logical entailment between the propositions corresponding to S and S_i.

3.2 Dempster-Shafer Theory

The belief logic of the previous section supposes that all available evidence has been pooled, so that evidence allotments $m(S)$ can be made on that basis. It also recognizes that the belief level of a given proposition may be underdetermined by available evidence. Dempster-Shafer theory [33, 34, 81] addresses the issue of how to combine different sources of evidence to obtain a single evidence allotment function $m(\cdot)$. It accomplishes this in such a way as to obtain point values for the function $m(\cdot)$, and therefore for any belief value $Bel(S_0)$ one wishes to deduce from the evidence base. The method rests on rather strong independence and normalization assumptions. However, one can drop the independence assumption and deduce intervals for $m(\cdot)$, rather than point values, by means of linear-fractional programming.

We again illustrate with an example. As before, the propositions of interest are x_1, x_2, and $x_1 \wedge x_2$, corresponding to sets S_1, S_2 and S_0, and we wish to deduce the level of evidence for $x_1 \wedge x_2$. Suppose that evidence source 1 determines an evidence allotment function $m_1(\cdot)$ by specifying $m_1(S_1) = 0.9$ and $m_1(\Theta) = 0.1$. Evidence source 2 specifies $m_2(S_2) = 0.8$ and $m_2(\Theta) = 0.2$. Then *Dempter's combination rule* derives a combined allotment function $\widehat{m}(\cdot) = m_1(\cdot) \otimes m_2(\cdot)$ by taking cross-products of evidence allotments to intersecting sets as shown in Table 1. This operation assumes that evidence sources are independent in some sense. Since none of the intersections are empty, we can set $m(\cdot) = \widehat{m}(\cdot)$; empty intersections require renormalization, as

Table 1 Example of Dempster's combination rule that does not require normalization.

	$m_2(S_2) = 0.8$	$m_2(\Theta) = 0.2$
$m_1(S_1) = 0.9$	$\widehat{m}(S_1 \cap S_2) = \widehat{m}(S_0) = 0.72$	$\widehat{m}(S_1 \cap \Theta) = \widehat{m}(S_1) = 0.18$
$m_1(\Theta) = 0.1$	$\widehat{m}(\Theta \cap S_2) = \widehat{m}(S_2) = 0.08$	$\widehat{m}(\Theta \cap \Theta) = \widehat{m}(\Theta) = 0.02$

Table 2 Example of Dempster's combination rule that requires normalization.

	$m_2(\overline{S}_1) = 0.25$	$m_2(\Theta) = 0.75$
$m_1(S_1) = 0.4$	$\widehat{m}(S_1 \cap \overline{S}_1) = \widehat{m}(\varnothing) = 0.1$	$\widehat{m}(S_1 \cap \Theta) = \widehat{m}(S_1) = 0.3$
$m_1(\Theta) = 0.6$	$\widehat{m}(\Theta \cap \overline{S}_1) = \widehat{m}(\overline{S}_1) = 0.15$	$\widehat{m}(\Theta \cap \Theta) = \widehat{m}(\Theta) = 0.45$

explained below. Because S_0 contains neither of the other sets, its belief value is simply $Bel(S_0) = m(S_0) = 0.72$. Note that this point value lies within the interval $[0.7, 0.8]$ that was derived for $Bel(S_0)$ in the previous section.

In general, we have

$$\widehat{m}(S) = m_1(S) \otimes m_2(S) = \sum_{\substack{A,B \\ S=A\cap B}} m_1(A)m_2(B)$$

If there are three evidence sources, one can compute $\widehat{m}(S)$ recursively as $\widehat{m}(S) = m_1(S) \otimes (m_2(S) \otimes m_3(S))$, and similarly for more numerous sources. If one or more intersections $A \cap B$ are empty, we must renormalize the evidence allotments. For example, the allotment functions shown in Table 2 assign positive evidence to contradictory propositions, since $\overline{S}_1$ is the complement of S_1. Since there can be no evidence for a logical contradiction, we assign $m(S_1 \cap \overline{S}_1) = m(\varnothing) = 0$. We now renormalize the remaining evidence allotments, yielding $m(S_1) = \frac{1}{3}$, $m(\overline{S}_1) = \frac{1}{6}$, and $m(\Theta) = \frac{1}{2}$. More generally,

$$m(S) = \begin{cases} \dfrac{\widehat{m}(S)}{1 - \widehat{m}(\varnothing)}, & \text{if } S \neq \varnothing \\ 0, & \text{if } S = \varnothing \end{cases}$$

In this manner, Dempster's combination rule can reconcile conflicting evidence sources, although one might question whether renormalization is a justifiable method for doing so. Finally, the belief value of S_0 is simply

$$Bel(S_0) = \sum_{S \subseteq S_0} m(S)$$

3.3 Belief Logic with Dempster's Combination Rule

The independence assumption in Dempster-Shafer theory can be dropped if we are content with inferred intervals rather than point values for the evidence allotment function $m(\cdot)$. This yields a belief logic similar to that described in Section 3.1, but with a mechanism for combining evidence sources. It also permits a linear programming model.

In the example of Table 1, we observe that

$$\begin{aligned}
&\widehat{m}(S_1 \cap S_2) + \widehat{m}(S_1) = 0.9 \\
&\widehat{m}(S_2) + \widehat{m}(\Theta) = 0.1 \\
&\widehat{m}(S_1 \cap S_2) + \widehat{m}(S_2) = 0.8 \\
&\widehat{m}(S_1) + \widehat{m}(\Theta) = 0.2 \\
&\widehat{m}(S_1), \widehat{m}(S_2), \widehat{m}(S_1 \cap S_2), \widehat{m}(\Theta) \geq 0
\end{aligned} \tag{23}$$

where any one of the four equations is redundant of the others. By minimizing and maximizing $\widehat{m}(S_1 \cap S_2)$ subject to (23), we obtain the range $[0.7, 0.8]$ for $\widehat{m}(S_1 \cap S_2)$, which contains the point value 0.72 obtained under an independence assumption. A similar approach works when renormalization is necessary. From Table 2, we have

$$\begin{aligned}
&\widehat{m}(\varnothing) + \widehat{m}(S_1) = 0.4 \\
&\widehat{m}(\overline{S}_1) + \widehat{m}(\Theta) = 0.6 \\
&\widehat{m}(\varnothing) + \widehat{m}(\overline{S}_1) = 0.25 \\
&\widehat{m}(S_1) + \widehat{m}(\Theta) = 0.75 \\
&\widehat{m}(S_1), \widehat{m}(\overline{S}_1), \widehat{m}(\varnothing), \widehat{m}(\Theta) \geq 0
\end{aligned} \tag{24}$$

Since $m(S_1) = \widehat{m}(S_1)/(1 - \widehat{m}(\varnothing))$, we can obtain a range for $m(S_1)$ by minimizing and maximizing this fraction subject to (24). This can be accomplished by fractional-linear programming, yielding the interval $[0.2, 0.4]$, which contains the point value $m(S_1) = \frac{1}{3}$ derived above.

In general, a range for $m(S_0)$, given nonempty S_0, is found as follows. Suppose that evidence source 1 provides positive evidence mass for sets $S_1, \ldots, S_m$, and evidence source 2 provides positive mass for $T_1, \ldots, T_n$. Then we wish to solve

$$\min/\max \left\{ \frac{\widehat{m}(S_0)}{1 - \widehat{m}(\varnothing)} \;\middle|\; \begin{array}{l} \sum_{j=1}^{n} \widehat{m}(S_i \cap T_j) = m_1(S_i), \;\; i = 1, \ldots, m \\ \sum_{i=1}^{m} \widehat{m}(S_i \cap T_j) = m_2(T_j), \;\; j = 1, \ldots, m \\ \widehat{m}(\cdot) \geq 0 \end{array} \right\}$$

This becomes a linear programming problem after the variable change $\mu(\cdot) = \widehat{m}(\cdot)t$:

$$\min/\max \left\{ \mu(S_0) \;\middle|\; \begin{array}{l} \sum_{j=1}^{n} \mu(S_i \cap T_j) = m_1(S_i)t, \;\; i = 1, \ldots, m \\ \sum_{i=1}^{m} \mu(S_i \cap T_j) = m_2(T_j)t, \;\; j = 1, \ldots, m \\ t - \mu(\varnothing) = 1; \quad \mu(\cdot), t \geq 0 \end{array} \right\}$$

When there are a large number of evidence sources, recursive computation of $m(S_0)$ can become difficult. The number of set intersections, and thus the number of sets with positive mass, can increase exponentially with the number of sources. Methods for simplifying the computations, using a set covering optimization model, are discussed in [24, 37]. In any event, it is likely that only a handful of evidence sources are relevant to a given conclusion in a given application.

A survey of algorithms for Dempster-Shafer theory can be found in [88], and a general discussion of semantics for belief logics in [42]. Other mathematical programming embeddings of logic, including predicate logic, are presented in [16, 24].

4 Nonmonotonic and Many-Valued Logics

This section deals with two nonstandard logics that have played a role in artificial intelligence, nonmonotonic logic and many-valued logic. It shows how optimization can make substantial contributions to their implementation, and even lend insight into the underlying ideas.

In *nonmonotonic logic*, the addition of new facts to a knowledge base may require the withdrawal of previously inferred conclusions. We show how integer programming can capture the semantics of default logic (a popular variety of nonmonotonic logic) and compute satisfying solutions.

Many-valued logic can assign truth values between true and false, values that may either be discrete or lie on a continuous scale. This permits a knowledge base to indicate the degree of confidence with which propositions can be asserted. Mixed integer/linear programming (MILP) is well suited to determine truth values of propositions deduced from the knowledge base.

4.1 Default Logic

In *default logic*, propositions with an "unless" clause are enforced only if the unless clause is not known to be true. This type of logic is closely related to answer set programming and negation-as-failure in logic programming. A key concept in the semantics of default logic is that of a *stable model* [40, 66]. Optimization can be helpful in this context because all stable models can be identified with the assistance of integer programming [9, 24].

Default logic is normally applied to a set of guarded rules having the form on the left:

$$P \rightarrow x_i \text{ unless } G, \quad \text{or} \quad \neg P \vee x_i \vee G$$

where P is a conjunction of zero or more atoms (atomic propositions) and logical formula G is the guard. The rule is semantically equivalent to the formula on the

right. One is permitted to use the rule $P \rightarrow x_i$ in a deduction as long as G is not known to be true.

In this context, a *model* for a set of guarded rules is a collection of atoms that, when set to true, satisfy the rules when the remaining atoms are presumed false. The concept of stable model can be explained with the following example. Consider the guarded rules on the left below, which are semantically equivalent to the formulas on the right.

$$\begin{array}{ll} x_1 \rightarrow x_2 \text{ unless } x_5 & \neg x_1 \vee x_2 \vee x_5 \\ x_1 \rightarrow x_3 \text{ unless } (x_2 \vee x_4) & \neg x_1 \vee x_3 \vee x_2 \vee x_4 \\ (x_1 \wedge x_4) \rightarrow x_5 & \neg x_1 \vee \neg x_4 \vee x_5 \\ \rightarrow x_1 \text{ unless } (x_4 \wedge x_5) & x_1 \vee (x_4 \wedge x_5) \end{array} \tag{25}$$

Note that a guard can be empty, in which case the rule is always enforced. A given model allows rules to be used when it fails to activate their guards. For example, the model $\{x_1.x_2\}$ admits the rules

$$\begin{array}{l} x_1 \rightarrow x_2 \\ (x_1 \wedge x_4) \rightarrow x_5 \\ \rightarrow x_1 \end{array} \tag{26}$$

where (26) is known as the *Gelfond-Lifschitz transform* of (25) with respect to the given model [40]. A *stable* model is one that minimally satisfies its Gelfond-Lifschitz transform; that is, flipping any atom from true to false would no longer satisfy the transform. It is easily checked that $\{x_1, x_2\}$ is a stable model.

Integer programming can identify all stable models by systematically finding all minimal models of the original rule base and checking which ones are minimal for the corresponding Gelfond-Lifschitz transform. A minimal model for (25), for example, is found by minimizing $\sum_i x_i$ subject to linear inequalities that encode the logical formulas in (25):

$$\min \left\{ \sum_i x_i \;\middle|\; \begin{array}{l} (1-x_1) + x_2 + x_3 \geq 1 \\ (1-x_1) + x_2 + x_3 + x_4 \geq 1 \\ (1-x_1) + (1-x_4) + x_5 \geq 1 \\ x_1 + x_4 \geq 1, \;\; x_1 + x_5 \geq 1 \\ x_i \in \{0,1\}, \text{ all } i \end{array} \right\} \tag{27}$$

Rules must be converted to conjunctive normal form before encoding, as illustrated by the last rule in (25), which is converted to $(x_1 \vee x_4) \wedge (x_1 \vee x_5)$. One optimal solution of (27) is $(x_1, \ldots, x_5) = (1, 1, 0, 0, 0)$, which corresponds to the model $\{x_1, x_2\}$, already found to be stable. Another minimal model can be found by adding the constraint $x_1 + x_2 \leq 1$ to exclude the model $\{x_1, x_2\}$, whereupon solution of (27) yields $(x_1, \ldots, x_5) = (0, 0, 0, 1, 1)$. This is not minimal in the G–L transform, which consists of the single rule $(x_1 \wedge x_4) \rightarrow x_5$. In fact, this rule is satisfied by flipping both ones to zero. Adding the further constraint $x_4 + x_5 \leq 1$ yields a minimal model $\{x_1, x_3, x_5\}$, which is not minimal in the corresponding G–L transform. Finally,

adding $x_1 + x_3 + x_5 \leq 2$ makes (27) infeasible. Thus (25) has no more minimal models, and $\{x_1, x_2\}$ is its only stable model.

Because the G–L transform is always a Horn system, unit propagation can check for minimality, with no need for integer programming at this stage. A Horn system consists of rules of the form $P \rightarrow x_i$, where P is a conjunction of zero or more atoms, and x_i may be absent (in which case $\neg P$ is asserted). There is exactly one minimal model (possibly empty) for a Horn system, which is found by applying unit propagation and noting which atoms are fixed to true. These atoms comprise the minimal model. Unit propagation proceeds by finding a rule in which P is empty, fixing its consequent x_i to true, and removing x_i from all other antecedents P in the rule base. This is repeated until no empty antecedents remain. For example, unit propagation applied to (26) first fixes x_1 to true, and then x_2, whereupon the rule $x_4 \rightarrow x_5$ remains. Thus $\{x_1, x_2\}$ is the unique minimal model of (26). Horn systems can also be solved by linear programming and have a number of interesting polyhedral properties [23, 55].

Subsequent research on integer programming methods for answer set programming and stable semantics includes [10, 64, 65, 77].

4.2 Many-valued Logic

Truth values in many-valued logic are frequently taken to be equally spaced numbers in the interval [0, 1], with larger values corresponding to greater degree of confidence. If desired, all real values in the interval [0, 1] can be regarded as truth values. Logical operators and connectives are semantically defined as functions of these truth values. For example, the truth value of $\neg P$ is $v(\neg P) = 1 - v(P)$, where $v(P)$ is the truth value of P. The truth value of the conditional $P \supset Q$ is frequently defined to be

$$v(P \supset Q) = \min\{1, 1 - v(P) + v(Q)\} \tag{28}$$

Mixed integer/linear programming (MILP) provides a natural mechanism for determining what truth values can inferred for a given formula, given premises with specified truth values [43]. MILP presupposes that the operators and connectives are *MILP-representable*, but a necessary and sufficient condition for representability is known, easily checked, and rather weak. In particular, Jeroslow [56] showed that an optimization problem is (finitely) MILP representable if and only if its feasible set[1] is the union of finitely many polyhedra that have the same recession directions. These are directions in which one can go forever without leaving the set. Thus $\boldsymbol{d}$ is a recession direction for set S if for some $\boldsymbol{v} \in S$, $\boldsymbol{v} + \alpha \boldsymbol{d} \in S$ for all $\alpha \geq 0$.

In classical logic, the truth value of a formula P is indicated by asserting P or $\neg P$. In multi-valued logic, one can specify (or bound) the truth value of P by means of a

[1] Technically, this condition applies to the epigraph rather than the feasible set, but the distinction is unnecessary here.

more general *signed formula*.[2] For example, the signed formula $\boxed{=0.6}\,P$ indicates that P has truth value 0.6, and $\boxed{\le 0.6}\,P$ indicates that P has some truth value in the interval $[0, 0.6]$.

To illustrate the role of MILP, assume for the moment that truth values are continuous in the interval $[0, 1]$. The resulting model is easily modified to allow only a finite number of discrete truth values. Suppose first that we wish to infer possible truth values of $P \supset Q$ from the premises $\boxed{\le p}\,P$ and $\boxed{\le q}\,Q$. For example, we may wish to know how false $P \supset Q$ can be, given the premises. We determine this by minimizing t subject to the condition that $\boxed{\le t}\,(P \supset Q)$ can be inferred from the premises. If t^* is the minimizing t, then $\boxed{\le t}\,(P \supset Q)$ can be inferred for all $t \in [t^*, 1]$, meaning that $P \supset Q$ cannot be more false than t^*. Using (28), the minimization problem is

$$\min_{t, v(P), v(Q)} \left\{ t \;\middle|\; t \ge \min\{1, 1 - v(P) + v(Q)\},\ 0 \le v(P) \le p,\ 0 \le v(Q) \le q \right\} \tag{29}$$

To obtain an MILP model of problem (29), we observe first that any feasible solution $(t, v(P), v(Q))$ must satisfy the following (inclusive) disjunction:

$$\begin{array}{lll} t \ge 1 & & t \ge 1 - v(P) + v(Q) \\ 0 \le v(P) \le v(Q) & \text{or} & 0 \le v(Q) \le q \\ 0 \le v(Q) \le q & & 0 \le v(P) \le p \end{array} \tag{30}$$

The feasible set is therefore the union of the two polyhedra described respectively by the two systems in (30), and illustrated respectively by Fig. 4(a) and 4(b). Each polyhedron has the single recession direction $(t, v(P), v(Q)) = (1, 0, 0)$, and the necessary and sufficient condition for MILP representability is therefore satisfied.

An MILP model is now obtained as follows. In general, it is shown in [57] that if the feasible set is a union of polyhedra described by $A^i \boldsymbol{x} \le \boldsymbol{b}^i$ for $i \in I$, the MILP constraint set consists of

$$\begin{array}{l} A^i \boldsymbol{x}^i \le \boldsymbol{b}^i \delta_i,\ i \in I \\ \boldsymbol{x} = \sum_{i \in I} \boldsymbol{x}^i,\ \ \sum_{i \in I} \delta_i = 1 \\ \delta_i \in \{0, 1\},\ i \in I \end{array}$$

where $\boldsymbol{x}^i$ and δ_i are new continuous and binary variables, respectively, for all $i \in I$. An MILP model of (29) therefore has the form

$$\min \left\{ t_1 + t_2 \;\middle|\; \begin{array}{ll} t_1 \ge \delta_1 & t_2 \ge \delta_2 - v_2(P) + v_2(Q) \\ 0 \le v_1(P) \le v_1(Q) & 0 \le v_2(Q) \le q\delta_2 \\ 0 \le v_1(Q) \le q\delta_1 & 0 \le v_2(P) \le p\delta_2 \\ \multicolumn{2}{c}{v(P) = v_1(P) + v_2(P)} \\ \multicolumn{2}{c}{v(Q) = v_1(Q) + v_2(Q)} \\ \multicolumn{2}{c}{\delta_1 + \delta_2 = 1,\ \delta_1, \delta_2 \in \{0, 1\}} \end{array} \right\} \tag{31}$$

[2] We slightly modify the notation in [43]

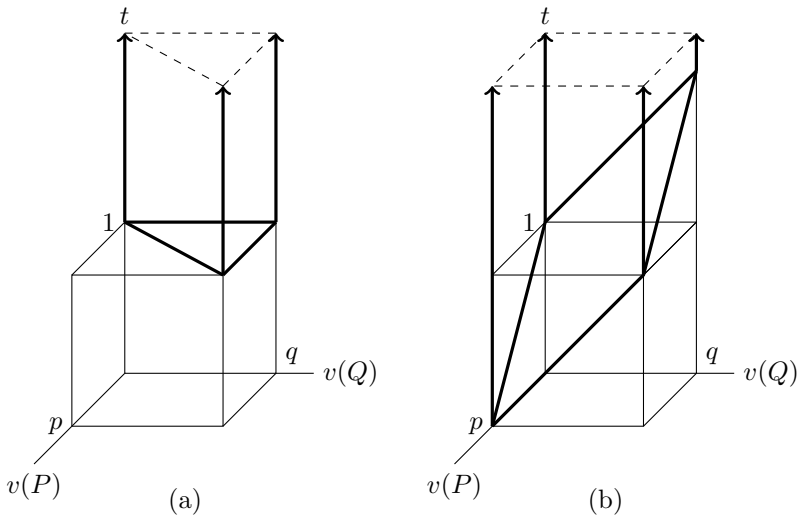

Fig. 4 Two polyhedra with the same unique recession direction $(v(P), v(Q), t) = (0, 0, 1)$. Heavy lines indicate the edges of the polyhedra.

It is straightforward (if somewhat tedious) to show that (31) simplifies to the MILP model

$$\min \left\{ t \;\middle|\; \begin{array}{ll} t \geq \delta & t \geq 1 - \delta - v(P) + v(Q) \\ 0 \leq v(P) \leq v(Q) + 1 - \delta & 0 \leq v(P) \leq p \\ 0 \leq v(Q) \leq q & \delta \in \{0, 1\} \end{array} \right\} \tag{32}$$

If we wish to recognize $m+1$ equally spaced discrete truth values in $[0, 1]$ rather than continuous truth values, we can replace $v(P)$ and $v(Q)$ with $v_I(P)/m$ and $v_I(Q)/m$, respectively in (32), and require that $v_I(P)$ and $v_I(Q)$ be integers. Then if t^* is the optimal solution of the resulting MILP problem, we can infer $\boxed{\leq t}(P \supset Q)$ for all $t \in \{t^*/m, (t^*+1)/m, \ldots, 1\}$.

The MILP model for more complex formulas can be constructed recursively. For example, the formula $(P \supset Q) \supset \neg P$ has the truth function.

$$v\big((P \supset Q) \supset \neg P\big) = \min\big\{1,\ 1 - \min\{1,\ 1 - v(P) + v(Q)\} + 1 - v(P)\big\}$$

An MILP model can be written by allowing the constraints in (32) to represent the possible truth values t of the inner implication $P \supset Q$, and observing that the truth value of $\neg P$ is $1 - v(P)$. Then we can write additional constraints to represent the possible truth values t' of the outer implication. Using δ' as the binary variable in the latter, this yields the model

$$\min\left\{ t' \;\middle|\; \begin{array}{ll} t' \geq \delta' & t' \geq 1-\delta' - t + (1-v(P)) \\ 0 \leq t \leq (1-v(P)) + 1-\delta' & 0 \leq t \leq 1, \quad \delta' \in \{0,1\} \\ \multicolumn{2}{c}{\text{constraints in (32)}} \end{array} \right\} \tag{33}$$

The valid bound $t \leq 1$ ensures that the polyhedra whose union (33) represents have the same recession direction $(t', t, v(P), v(Q)) = (1, 0, 0, 0)$.

As an example, suppose the upper bounds on $v(P)$ and $v(Q)$ are $(p, q) = (0.3, 0.7)$. The minimum value of t' is 0.7, indicating that $(P \supset Q) \supset \neg P$ cannot be more false than 0.7. This solution is achieved when $(t, v(P), v(Q)) = (1, 0.3, 0.3)$ and $(\delta, \delta') = (0, 0)$. A branch-and-bound solution of this problem instance appears in Section 7.2.

Additional discussion of MILP models for multivalued logic can be found in [24, 43].

5 Statistical Inference of Logical Formulas

As noted in the introduction, the estimation of weights in a neural network poses an optimization problem. The same is true for the calibration of a vector support machine, which is a hyperplane that classifies by separating two (possibly overlapping) clusters of data points. There is a third data fitting possibility that is often overlooked. One can fit a *logical formula* to data in a manner analogous to classical regression, which likewise gives rise to an optimization problem. This represents yet another instance in which logic and optimization combine to serve the purposes of AI. Like logic-based methods in general, it has the advantage of transparency, since one can examine precisely what rules are inferred from training data. It also offers the possibility of computing the statistical significance of an inferred formula, as is routinely done in regression analysis. A similar approach can be used to reduce a trained neural network to a logical formula that approximates its predictions, for the sake of transparency, by fitting the formula to the output of the network for various inputs.

5.1 Boolean Regression

Boolean regression [19] infers a logical formula from boolean data points of the form $(\boldsymbol{x}, y)$, where the independent variables $\boldsymbol{x} = (x_1, \ldots, x_m)$ represent observed attributes and the dependent variable y represents the observed outcome. Following classical regression analysis, it supposes there is noise in the observations and therefore writes

$$y = f(\boldsymbol{x}) \oplus \epsilon$$

where $f(\boldsymbol{x})$ is the true outcome, and ϵ is the observation error. Here, $\oplus$ is a binary sum, so that $a \oplus b = (a + b) \bmod 2$. The boolean error term ϵ is a simple Bernoulli

variable that takes the value 1 with probability p and 0 with probability $1 - p$. The value of p is estimated along with the regression formula, much as the error variance is estimated along with the regression coefficients in classical statistics. An independent variable x_i that is nonboolean and takes discrete values $0, 1, 2, \ldots, k$ can be accommodated by replacing it with boolean variables $x_{i0}, \ldots, x_{i\ell}$, where $x_i = x_{i0} + 2x_{i1} + \cdots + 2^\ell x_{i\ell}$.

The dataset may contain multiple observations of y for a given value of $\boldsymbol{x}$, and no observation of y at all for other values of $\boldsymbol{x}$. This is illustrated by an example taken from [19], whose dataset (Table 3) is designed to contain a great deal of noise. We wish to identify a boolean function $f(\boldsymbol{x})$ that best fits the data, where f belongs to a specified class F of boolean functions, such as the class of functions expressed by a boolean formula of a certain form.

Table 3 Small dataset for boolean regression.

Observations of $\boldsymbol{x}$					Number of observations with	
x_1	x_2	x_3	x_4	x_5	$y = 0$	$y = 1$
0	1	0	1	1	7	15
1	1	0	0	1	9	3
1	0	0	1	1	8	2
1	0	0	0	1	3	7
1	1	0	1	0	5	17
0	1	1	1	1	9	2

A classical least-squares fit is based on maximum likelihood estimation, and a similar approach can be used here. We seek a function f that maximizes the likelihood of the errors displayed in Table 3. Thus if $\boldsymbol{y} = (y_1, \ldots, y_n)$ represents the set of observations, we wish to identify a function $f \in F$ and probability p that maximize the likelihood function

$$\Pr(\boldsymbol{y}|f, p) = (1 - p)^{n-e(f)} p^{e(f)} \tag{34}$$

where n is the number of observations, and $e(f)$ is the number of errors made by function f on the observed values of $\boldsymbol{x}$. For example, if $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^6)$ are the observed values of $\boldsymbol{x}$ in Table 3, and $(f(\boldsymbol{x}^1), \ldots, f(\boldsymbol{x}^6)) = (0, \ldots, 0)$, then $Pr(\boldsymbol{y}|f, p) = (1 - p)^{41} p^{46}$. If $(f(\boldsymbol{x}^1), \ldots, f(\boldsymbol{x}^6)) = (0, 0, 0, 0, 0, 1)$, then $Pr(\boldsymbol{y}|f, p) = (1 - p)^{34} p^{53}$, and so forth.

We can note right away that (34) is maximized with respect to p when $p = e(f)/n$. It therefore suffices to maximize the following over $f \in F$:

$$\Pr\big(\boldsymbol{y}\big|f, e(f)/n\big) = \Big(1 - \frac{e(f)}{n}\Big)^{n-e(f)} \Big(\frac{e(f)}{n}\Big)^{e(f)} \tag{35}$$

Interestingly, since (35) is convex in $e(f)$, it is maximized either by a function f that minimizes $e(f)$ or by a function that maximizes $e(f)$. Most applications call for a fit that minimizes errors, but there are cases in which it may be appropriate to

maximize errors [19]. For present purposes, we assume that the data are such that a maximum likelihood fit is error minimizing. If $\hat{f}$ is the error minimizing function, then $\hat{p} = e(\hat{f})/n$ is a (biased) estimator of p.

Suppose now that we wish to fit to the data of Table 3 a regression formula that is a logical clause with positive literals:

$$\beta_0 \vee \beta_1 x_1 \vee \cdots \vee \beta_5 x_5 \tag{36}$$

The binary coefficient $\beta_i = 1$ (for $i \geq 1$) indicates that x_i is present in the clause, and $\beta_i = 0$ indicates that x_i is absent. The clause is a tautology if $\beta_0 = 1$, and β_0 disappears from the clause if $\beta_0 = 0$. The task is to find a maximum likelihood estimate of $\beta_0, \ldots, \beta_5$, which is accomplished by maximizing (35) over all functions f that result from some setting of $\beta_0, \ldots, \beta_5 \in \{0, 1\}$.

The maximization problem can be solved by formulating it as a pseudoboolean optimization problem, mentioned earlier in the context of probabilistic logic. Let $f_{\boldsymbol{\beta}}$ be the boolean function that results from a given tuple $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_5)$ of parameters in (36). We first note that if

$$\beta_0 = \beta_2 = \beta_4 = \beta_5 = 0 \tag{37}$$

then $f_{\boldsymbol{\beta}}(\boldsymbol{x}^1) = 0$, and line 1 of Table 3 generates 15 errors. If (37) does not hold, then $f_{\boldsymbol{\beta}}(\boldsymbol{x}^1) = 1$, and there are 7 errors. So, line 1 generates

$$15\bar{\beta}_0\bar{\beta}_2\bar{\beta}_4\bar{\beta}_5 + 7(1 - \bar{\beta}_0\bar{\beta}_2\bar{\beta}_4\bar{\beta}_5)$$

errors, where $\bar{\beta}_i = 1 - \beta_i$. Summing expressions of this sort for $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^6$ and collecting terms, we obtain the pseudoboolean function $e(f_{\boldsymbol{\beta}})$ equal to

$$41 + 8\bar{\beta}_0\bar{\beta}_2\bar{\beta}_4\bar{\beta}_5 - 6\bar{\beta}_0\bar{\beta}_1\bar{\beta}_2\bar{\beta}_5 - 6\bar{\beta}_0\bar{\beta}_1\bar{\beta}_4\bar{\beta}_5 + 4\bar{\beta}_0\bar{\beta}_1\bar{\beta}_5 + 12\bar{\beta}_0\bar{\beta}_1\bar{\beta}_2\bar{\beta}_4 - 7\bar{\beta}_0\bar{\beta}_3\bar{\beta}_4\bar{\beta}_5$$

The minimum error solution is $\boldsymbol{\beta} = (0, 0, 1, 0, 0, 0)$, with $e(f_{\boldsymbol{\beta}}) = 32$ errors, which means the estimated error probability is $\hat{p} = e(f_{\boldsymbol{\beta}})/n = 32/87 \approx 0.368$. This solution corresponds to the simple logical formula x_2, which ignores all but one of the independent variables. To obtain a more satisfactory formula, one might allow negative as well as positive literals:

$$\beta_0 \vee \beta_1 x_1 \vee \cdots \vee \beta_5 x_5 \vee \gamma_1(\neg x_1) \vee \cdots \vee \gamma_5(\neg x_5) \tag{38}$$

The best-fit formula is $x_2 \vee \neg x_4$, which results in $e(f_{\boldsymbol{\beta},\boldsymbol{\gamma}}) = 28$ errors and $\hat{p} \approx 0.322$.

In general, the regression formula can be any expression of the form

$$f(\boldsymbol{x}_\beta) = \bigvee_{i \in I} \beta_i P_i(\boldsymbol{x}) \tag{39}$$

where each $P_i(\boldsymbol{x})$ is any desired logical proposition that contains atomic propositions in $\boldsymbol{x} = (x_1, \ldots, x_m)$. For example, each $P_i(\boldsymbol{x})$ could be a conjunction of literals (atomic propositions or their negations); in fact, any propositional formula can

be written as (39) with $P_i(\boldsymbol{x})$s in this form. If $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n$ are the observations of independent variables, let $y_0(\boldsymbol{x}^j)$ be the number of observations of $f(\boldsymbol{x}^j)$ for which $y = 0$, and $y_1(\boldsymbol{x}^j)$ the number for which $y = 1$. The pseudoboolan function to be minimized is

$$e(f_{\boldsymbol{\beta}}) = \sum_{j=1}^{n} y_0(\boldsymbol{x}^j) + \sum_{j=1}^{n} \left(y_1(\boldsymbol{x}^j) - y_0(\boldsymbol{x}^j)\right) \prod_{i \,|\, P_i(\boldsymbol{x}^j)=1} \bar{\beta}_i$$

Solution methods for pseudoboolean optimization are extensively discussed in [18]. Recent developments in solution methods include [38, 39, 83], and schemes for integrating MILP techniques are described in [35, 36].

5.2 Statistical Significance

Boolean regression can provide meaaures of statistical significance, including confidence levels and the significance of regression, the latter of which enables stepwise regression. A potential barrier to calculating significance is that the distribution of estimated regression parameters depends on the distribution of observational error, which is unknown *a priori*. Classical regression deals with this problem by identifying a "pivot" statistic whose distribution depends only on the number of data points (or more precisely, the degrees of freedom). For example, the t-statistic allows one to compute confidence intervals based on the pre-computed distribution of the t-statistic. Apparently, no pivot statistic has been identified for boolean regression, but one can nonetheless obtain useful significance indicators from a Bayesian model. We follow here the analysis of [19].

A Bayesian significance model identifies the confidence level of a deduced regression formula f with the posterior probability that f is the true formula, given the observed data and prior probabilities. Thus, given observations $\boldsymbol{y} = (y_1, \ldots, y_n)$, the confidence level of f is $\Pr(f|\boldsymbol{y})$. Using Bayes' rule, this is

$$\Pr(f|\boldsymbol{y}) = \frac{\Pr(\boldsymbol{y}|f)\Pr(f)}{\Pr(\boldsymbol{y})} \tag{40}$$

where $\Pr(f)$ is the prior probability that function f is the correct one. The conditional probability $\Pr(\boldsymbol{y}|f)$ is obtained by integrating over possible error probabilities p:

$$\Pr(\boldsymbol{y}|f) = \int_0^1 \Pr(\boldsymbol{y}|f, p)\pi(p)dp \tag{41}$$

where $\pi(p)$ is a prior probability density function for p, and $\Pr(\boldsymbol{y}|f, p)$ is given by (34). Also $\Pr(\boldsymbol{y})$ is

$$\Pr(\boldsymbol{y}) = \sum_{f' \in F} \Pr(\boldsymbol{y}|f')\Pr(f')$$

Since nothing is known in advance about which formula could be correct, we assume $\Pr(f')$ is the same for all $f' \in F$ and can therefore ignore this term. As for the probability p of error, we can suppose it is no greater than some value $\rho \leq \frac{1}{2}$, since even a random guess is right half the time. Since we know nothing beyond this, we assume p is uniformly distributed over $[0, \rho]$, with $\pi(p) = 1/\rho$ in this range. Thus, using (34), (41) becomes

$$\Pr(Y|f) = \frac{1}{\rho} \int_0^\rho (1-p)^{n-e(f)} p^{e(f)} dp$$

Evaulating the integral, we obtain

$$Pr(\boldsymbol{y}|f) = \frac{\rho^{e(f)}}{n+1} \sum_{i=1}^{n-e(f)} \binom{n-e(f)}{i} \binom{n}{i}^{-1} (1-\rho)^{n-e(f)-i} \tag{42}$$

It is reported in [19] that, using this analysis, the maximum likelihood formula x_2 can be inferred from the data in Table 3 with confidence level 0.282 when $\rho = \frac{1}{2}$. This says one cannot have much confidence that the formula is exactly right. One might calculate a confidence sphere that is analogous to an confidence interval in classical regression. A confidence sphere could consist of all binary vectors $\boldsymbol{\beta}$ within a Hamming distance of d from the inferred vector $\boldsymbol{\beta}$ of coefficients. In the example, the confidence level increases to 0.433 when $d = 1$ and 0.572 when $d = 2$, still rather low due to the large amount of noise in the dataset.

The significance of regression, however, is more encouraging. We can compute it by considering the hypotheses:

$$\begin{aligned} H_0 &: \ \beta_i = 0 \text{ for } i = 1, \ldots, 5 \text{ (null hypothesis)} \\ H_1 &: \ \beta_i = 1 \text{ for at least one } i \in \{1, \ldots, 5\} \end{aligned}$$

The best fit under the null hypothesis is the function f_0 defined by $\beta_0 = 1$, with $\Pr(f_0|\boldsymbol{y}) = 0.0104$. The best fit under H_1 is again the function f corresponding to $\boldsymbol{\beta} = (0, 0, 1, 0, 0, 0)$ with $\Pr(f|\boldsymbol{y}) = 0.278$. The significance of regression is

$$\frac{\Pr(f_0|\boldsymbol{y})}{\Pr(f_0|\boldsymbol{y}) + \Pr(f|\boldsymbol{y})} = 0.036$$

This means that it is very likely (96.4% probability) that an expression of the form $\beta_1 x_1 \vee \cdots \vee \beta_5 x_5$ captures a relationship that is not purely the result of chance.

We can also perform stepwise regression by adding negative literals to the formula as in (38). This significantly improves the fit if we can reject H_0:

$$\begin{aligned} H_0 &: \ \beta_i = 0 \text{ for } i = 6, \ldots, 10 \text{ (i.e., the negative literals do not improve the fit)} \\ H_1 &: \ \beta = 1 \text{ for at least one } i \in \{6, \ldots, 10\} \end{aligned}$$

We can reject H_0 with probability $1 - 0.073 = 0.927$, which means that the improvement in fit is significant at the 10% level but not at the 5% level. In addition,

the confidence level of the expanded regression formula $x_2 \vee \neg x_4$ is only 0.11, because it is competing with a much larger collection of admissible formulas.

Related research on learning boolean functions appears in [8, 26, 70, 71, 79, 82]. An important class of boolean functions are monotone, meaning that $\boldsymbol{x}' \geq \boldsymbol{x}$ implies $f(\boldsymbol{x}') \geq f(\boldsymbol{x})$, where $\boldsymbol{x}' \geq \boldsymbol{x}$ when $x'_j \geq x_j$ for each j. For example, a monotone boolean function implies that observing additional indicators of a particular disease cannot reverse a prediction that the disease is present. The problem of learning monotone Boolean functions from noisy and incomplete data is addressed in [2, 13, 17].

6 Inference as Projection

Optimization and logical inference are fundamentally related because they are both special cases of projection [51]. The *projection* of a set S of tuples $\boldsymbol{x} = (x_1, \ldots, x_n)$ onto variables $\bar{\boldsymbol{x}} = (x_1, \ldots, x_k)$ is

$$S|_{\bar{\boldsymbol{x}}} = \left\{(x_1, \ldots, x_k) \;\middle|\; (x_1, \ldots, x_n) \in S\right\}$$

Optimization is a special case of projection because any optimization problem can be written $\min/\max\{z \mid (z, \boldsymbol{x}) \in S\}$ and solved by computing the projection $S|_z$. The minimum and maximum values of z, if they exist, are then easily recognized as the smallest and largest elements of $S|_z$. Inference can be seen as a special case of projection if we let $S(F)$ denote the set of assignments to $\boldsymbol{x}$ that satisfy a set F of logical formulas. Then a formula f containing variables $\bar{\boldsymbol{x}}$ can be inferred from F if and only if all the assignments in the projection $S(F)|_{\bar{\boldsymbol{x}}}$ satisfy f.

This linkage of optimization and inference through projection provides a novel opportunity to apply optimization to inference. The projection of $S(F)$ onto $\bar{\boldsymbol{x}}$ can be obtained with the assistance of a binary decision diagram (BDD), which compactly and transparently displays satisfying solutions of F. The projection can be computed using a generalization of Benders decomposition, a well-known optimization method that, in effect, (partially) computes the projection of the feasible set onto a subset of variables. The first subsection below illustrates how a BDD represents the satisfying set $S(F)$, and how it relates to projection. The second subsection shows how Benders decomposition can accelerate the computation of the projection, thereby deducing all formulas that can be inferred from F.

This methodology can be useful to AI because it allows one to infer everything that can be deduced from a knowledge base regarding a specific topic, rather than check which individual formulas can be deduced. For example, one may be interested in whether substances A, B, and C can interact to cause allergic reactions D and E. Rather than check whether one can deduce that A and B interact to cause D (i.e. $(A \wedge B) \supset D$), whether one can deduce that B and C cause reaction E, and so forth, one can project the knowledge base onto the five variables A, B, C, D, and E. Since one is projecting onto a handful of variables, the projection can be computed fairly

rapidly. The rather small number of assignments in the projection (at most 2^5) is easily converted to a few logical formulas that capture everything that is known about allergic reactions to A, B and C.

6.1 Projection in a Binary Decision Diagram

Binary decision diagrams were introduced in the 1970s and have been used for logic circuit verification, product configuration, and more recently, discrete optimization [1, 12, 14, 21, 78]. We are interested in their potential for computing projections. A BDD is a graphical representation of a boolean function $f(\boldsymbol{x})$. Any set F of propositional formulas defines a boolean function $f(\boldsymbol{x})$ that takes the value 1 when $\boldsymbol{x}$ satisfies F and 0 otherwise, where $\boldsymbol{x} = (x_1, \dots, x_n)$ are the variables in F. Thus, any set F of formulas is represented by a suitable BDD.

Consider, for example, the set F of formulas

$$\begin{array}{ll} x_1x_2 \supset (x_3\bar{x}_4x_5\bar{x}_6) & (f_1) \\ \bar{x}_1\bar{x}_2 \supset (\bar{x}_4x_5\bar{x}_6 \vee x_3x_4x_5x_6) & (f_2) \\ (x_1\bar{x}_2 \vee \bar{x}_1x_2) \supset \bar{x}_3\bar{x}_4\bar{x}_5x_6 & (f_3) \end{array} \tag{43}$$

where, for readability, x_ix_j means $x_i \wedge x_j$ and $\bar{x}_i$ means $\neg x_i$. The set F is represented by the BDD in Fig. 5(a). The nodes of the BDD are arranged in layers that, except for the terminal node at the bottom, correspond to variables $x_1, \dots, x_6$. A dashed arc leaving a node in layer i toward a layer below it represents setting $x_i = 0$, and a solid arc represets $x_i = 1$. Each path from top to bottom therefore represents an assignment to $\boldsymbol{x}$. The BDD is constructed so that the top-to-bottom paths correspond exactly to assignments that satisfy the formulas in F.[3] The BDD in Fig. 5(a) is *reduced* in the sense that no smaller BDD that uses the same variable ordering represents (43). In fact, any given boolean function is uniquely represented by a reduced BDD with a specified variable ordering [21]. Methods for constructing reduced BDDs are described in [86, 87].

Let's suppose that we wish to know what can be deduced from (43) regarding the variables x_2, x_4, and x_6. To deduce all formulas containing x_2, x_4, x_6 that are implied by (43), we can derive the BDD B' that results from B by projecting out variables x_1, x_3, x_5. Then the top-to-bottom paths in B' correspond to the assignments in the projection onto (x_2, x_4, x_6). The deducible formulas are then easily obtained from this projection, as we will illustrate shortly.

The classical method for projecting out a variable x_i proceeds by first obtaining the BDDs B_1 and B_0 that result from setting x_i to 1 and 0, respectively, and then computing B' by taking the disjunction of B_1 and B_0. However, the disjunction operation on BDDs tends to be quite expensive, and it must be performed recursively for each variable to be projected out.

[3] Classically, BDDs also contain paths to a second terminal node that represent non-satisfying assignments, but these can be omitted for present purposes.

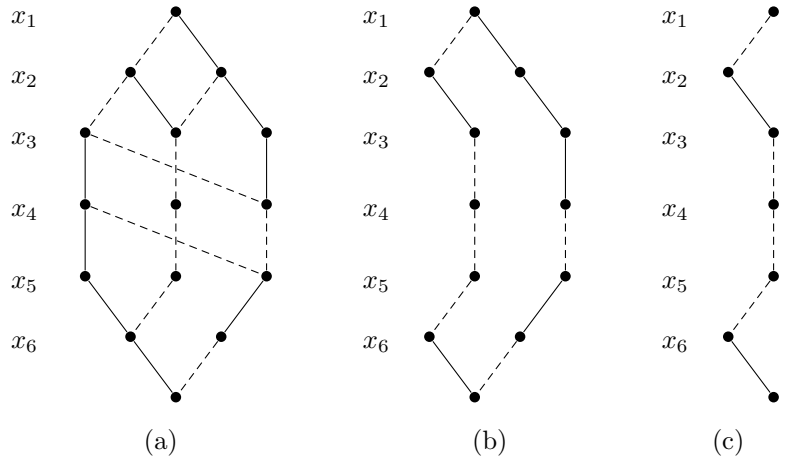

Fig. 5 (a) Reduced binary decision diagram for a small set of logical propositions. (b) Reduced BDD after fixing $x_2 = 1$. (c) Reduced BDD after fixing $(x_2, x_4, x_6) = (1, 0, 1)$.

Since we wish to project (43) onto a small number of variables, it is more efficient to check directly which assignments to (x_2, x_4, x_6) are represented by top-to-bottom paths in B. Suppose, for example, we first check $(x_2, x_4, x_6) = (1, 1, 1)$. We might begin by setting $x_2 = 1$ and removing the dashed arcs corresponding to $x_2 = 0$ in Fig. 5(a). This permits the removal of several arcs and nodes because they no longer lie on a top-to-bottom path, resulting in the smaller BDD of Fig. 5(b). Now if we try setting $x_4 = 1$ and remove all dashed arcs corresponding to $x_4 = 0$, this causes all remaining arcs to be removed. Since this already eliminates all top-to-bottom paths, (x_2, x_4, x_6) can be neither $(1, 1, 0)$ nor $(1, 1, 1)$. Now if we try $(x_2, x_4, x_6) = (1, 0, 1)$, the BDD reduces to that in Fig. 5(c). Since there is a top-to-bottom path (in this case, a single such path), we conclude that $(1, 0, 1)$ is part of the projection. Continuing in this fashion, we find that the projection onto (x_2, x_4, x_6) is

$$\big\{(1, 0, 1), (1, 0, 0), (0, 1, 1), (0, 0, 1), (0, 0, 0)\big\} \tag{44}$$

It is readily seen that these are the satisfying assignments of the formulas $x_2 \supset \neg x_4$ and $x_4 \supset x_6$. These formulas therefore represent everything that can be deduced from (43) regarding variables x_2, x_4, x_6.

A possible barrier to the use of reduced BDDs is their ability to grow exponentially with the number of variables. However, the size of the BDD depends on other factors as well. For example, the knowledge base may decouple into sections that deal with different topics and therefore have no variables in common. In such cases, a BDD can be generated for each section independently of the others, resulting in much smaller diagrams. A generalization of this approach exploits the structure of the dependency graph of the knowledge base. The dependency graph contains a vertex for every formula, and two vertices are connected by an edge when the corresponding formulas have at least one variable in common. The complexity of logical inference

is at worst exponential in the treewidth of the dependency graph, which tends to be small when the formulas are loosely coupled. In such cases, one can build a nonserial BDD as described in [50], which can be markedly smaller than a conventional serial BDD and is quite suitable for computing projections. In addition, the size of a serial or nonserial BDD can be very sensitive to the ordering of the variables, and it may be possible to find an ordering that results in a much smaller diagram [86, 87]. Finally, given that AI's large language models are based on neural networks that may contain upwards of a trillion parameters, a very large BDD may be acceptable in AI applications.

6.2 Projection with Benders Decomposition

The previous section showed how to compute the projection of (43) onto (x_2, x_4, x_6) by employing a decision diagram to check every possible assignment to these variables for consistency with (43). It is generally possible, however, to accelerate this process significantly by applying Benders decomposition. This popular optimization technique was originally applied to mixed integer programming [11], but we use a generalization known as logic-based Benders decomposition (LBBD) [49, 52, 53].

LBBD is applied to a general optimization problem of the form

$$\max \left\{ f(\boldsymbol{x}, \boldsymbol{y}) \;\middle|\; (\boldsymbol{x}, \boldsymbol{y}) \in S,\ \boldsymbol{x} \in D_{\boldsymbol{x}},\ \boldsymbol{y} \in D_{\boldsymbol{y}} \right\}$$

where $D_{\boldsymbol{x}}$ and $D_{\boldsymbol{y}}$ are the domains of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively (e.g., tuples of integers or reals). For our purposes, we can restrict attention to problems in which the objective function depends only on $\boldsymbol{x}$:

$$\max \left\{ f(\boldsymbol{x}) \;\middle|\; (\boldsymbol{x}, \boldsymbol{y}) \in S,\ \boldsymbol{x} \in D_{\boldsymbol{x}},\ \boldsymbol{y} \in D_{\boldsymbol{y}} \right\} \tag{45}$$

The problem is decomposed into a master problem

$$\max \left\{ f(\boldsymbol{x}) \;\middle|\; \text{Benders cuts},\ \boldsymbol{x} \in D_{\boldsymbol{x}} \right\}$$

and a feasibility subproblem that asks whether $(\bar{\boldsymbol{x}}, \boldsymbol{y}) \in S$ for some $\boldsymbol{y} \in D_{\boldsymbol{y}}$, where $\bar{\boldsymbol{x}}$ is the most recent solution of the master problem. If the subproblem has a feasible solution $\bar{\boldsymbol{y}}$, the original problem (45) has optimal solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$. If the subproblem is infeasible, the proof of infeasibility is analyzed to obtain a valid constraint (Benders cut) $\boldsymbol{x} \in S_{\bar{\boldsymbol{x}}}$ that is violated by $\boldsymbol{x} = \bar{\boldsymbol{x}}$ as well as by other values of $\boldsymbol{x}$ that the same proof shows to be infeasible. The Benders cut is added to the constraint set of the master problem, which is re-solved to obtain the next $\bar{\boldsymbol{x}}$. The process continues until (a) the subproblem is feasible, or (b) the master problem is infeasible (in which case the original problem infeasible).

Of greatest interest to us is the fact that Benders cuts generated during this process at least partially define the projection of the feasible set of (45) onto $\boldsymbol{x}$. A complete description of the projection can be obtained simply by adding the constraint $\boldsymbol{x} \neq \bar{\boldsymbol{x}}$

to the master problem (and continuing to generate cuts) each time an optimal solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is found. When the master problem finally becomes infeasible, the accumulated Benders cuts completely describe the projection onto $\boldsymbol{x}$.

To project (43) onto (x_2, x_4, x_6), we solve the problem

$$\max \left\{ x_2 + x_4 + x_6 \;\middle|\; (43) \right\} \tag{46}$$

by LBBD as follows.[4] We let $\boldsymbol{x} = (x_2, x_4, x_6)$ and $\boldsymbol{y} = (x_1, x_3, x_5)$, so that the master problem is the integer programming problem

$$\max \left\{ x_2 + x_4 + x_6 \;\middle|\; \text{Benders cuts}, \;\; x_2, x_4, x_6 \in \{0, 1\} \right\} \tag{47}$$

Initially there are no Benders cuts, and the solution is $(x_2, x_4, x_6) = (1, 1, 1)$. This defines a subproblem that seeks a feasible solution of (46) in which $(x_2, x_4, x_6) = (1, 1, 1)$. That is, we seek top-to-bottom path in the BDD of Fig. 5(a) in which arcs corresponding to $(x_2, x_4, x_6) = (0, 0, 0)$ are removed. We solved this subproblem in the previous section by first fixing $x_1 = 1$ and generating the BDD of Fig. 5(b). At this point we noted that $(x_2, x_4) = (1, 1)$ is already infeasible. This allows us to create a Benders cut that excludes solutions in which $(x_2, x_4) = (1, 1)$. We write the cut as $(1 - x_2) + (1 - x_4) \geq 1$ and add it to the master problem (47). Re-solving the master problem, we obtain $(x_2, x_4, x_6) = (1, 0, 1)$, which defines a feasible subproblem and therefore solves (46). To continue generating Benders cuts, we add the constraint $(x_2, x_4, x_6) \neq (1, 0, 1)$ to the master problem by writing $(1 - x_2) + x_4 + (1 - x_6) \geq 1$. After generating the optimal solutions (44) in this fashion, the master problem becomes infeasible, and we have the projection onto (x_2, x_4, x_6). In larger instances, the Benders cuts speed up this process by eliminating many infeasible assignments to the master problem variables before they are submitted to the subproblem.

7 Transparency through Postoptimality Analysis

Optimization-based inference can contribute significantly to transparency in AI applications by means of postoptimality analysis. This type of analysis can identify constraints that are essential to proving optimality, and in many cases, whether or how much the optimal solution would change if the constraints were modified a given amount. When applied to inference, postoptimality analysis can identify premises in the knowledge base that play a role in deriving inferred propositions, and perhaps determine whether and how much the inferred propositions would change if the premises were modified.

We begin with a brief discussion of postoptimality analysis in linear programming, which we applied in earlier sections to probabilistic logic and various belief logics. We then describe two methods of postoptimality analysis for mixed integer/linear programming (MILP), which we applied to nonmonotonic logic, multivalued logic,

[4] We could just as well minimize, or replace any x_i by $-x_i$.

boolean regression, and inference via projection. One is an inference-based method that uses dual multipliers obtained in the branch-and-bound search tree. The other is based on a BDD that encodes near-optimal solutions of the problem.

7.1 Postoptimality Analysis in Linear Programming

Postoptimality analysis in linear programming (LP) consists primarily of elementary techniques that have been implemented in LP solvers for decades. An LP problem has the form

$$\min \left\{ \boldsymbol{c}^\top \boldsymbol{x} \;\middle|\; A\boldsymbol{x} \geq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0} \right\} \tag{48}$$

The corresponding *dual* problem is

$$\max \left\{ \boldsymbol{u}^\top \boldsymbol{b} \;\middle|\; \boldsymbol{u}^\top A \leq \boldsymbol{c}^\top,\ \boldsymbol{u} \geq \boldsymbol{0} \right\}$$

The dual variables $\boldsymbol{u} = (u_1, \ldots, u_m)$ correspond to the m constraints $A\boldsymbol{x} \geq \boldsymbol{b}$ of the original problem (48). Under weak conditions, an LP problem has the same optimal value z^* as its dual, and the dual solution is obtained for free as a byproduct of solving the original problem.

The optimal dual solution $\boldsymbol{u}$ is a key to postoptimality analysis. The dual multiplier u_i associated with constraint i of $A\boldsymbol{x} \geq \boldsymbol{b}$ tells us that if the right-hand side b_i of constraint i is changed to $b_i + \Delta b_i$ (where Δb_i can be negative), the resulting optimal value is at least $z^* + u_i \Delta b_i$. Also, one can easily compute a range of perturbations Δb_i within which the new optimal value is *exactly* $z^* + u_i \Delta b_i$. A consequence of this is that only constraints with positive dual multipliers u_i serve as premises in the proof of optimality, and all other constraints can be dropped without changing the optimal solution. In fact, it is not hard to state conditions under which this set of premises is irreducible, meaning that they are all essential to the proof. This and related matters are studied in [27, 28, 29, 30].

As an example, consider the probabilistic logic problem instance discussed in Section 2.1, in which probablities 0.9, 0.8, and 0.7 are assigned to x_1, $x_2 \supset x_2$, and $x_2 \supset x_3$, respectively. The inferred probability range for x_3 is $[0.5, 0.7]$. The lower limit 0.5 is obtained by minimizing $\Pr(x_3)$, and the resulting dual multipliers for the three probability assignments are all 1.25. So, reducing the assigned probability 0.9 to 0.8, for example, results in new a lower limit of at least $0.5 - (1.25)(0.1) = 0.375$. In fact, the lower limit is exactly 0.375, because this calculation is exact for any perturbation $\Delta b_1 \in [-0.4, 0]$. As it happens any positive perturbation results in an infeasible problem, and similarly for the other two assigned probabilities.[5] The upper limit 0.7, which results from maximizing $\Pr(x_3)$, yields dual multipliers 0, 0, and 1 for the three assigned probabilities. This means that only the proposition $x_2 \supset x_3$

[5] The calculated value remains a valid lower bound on the optimal value, because the optimal value of an infeasible minimization problem is ∞.

plays a role in inferring the upper limit on $\Pr(x_3)$, and the other two probability assignments can be dropped without affecting it.

7.2 Inference-Based MILP Postoptimality Analysis

An inference-based method [32] provides comprehensive postoptimality analysis for MILP when the problem has moderate size, and a limited analysis for larger instances. We will suppose the integer-valued variables in the problem are binary, in which case it has the form

$$\min \left\{ z = \boldsymbol{c}^\top \boldsymbol{x} \;\middle|\; A\boldsymbol{x} \geq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0},\ x_i \in \{0,1\} \text{ for } i \in I \right\} \tag{49}$$

It is convenient to suppose that $A\boldsymbol{x} \geq \boldsymbol{b}$ contains constraints $x_i \leq 1$ for $i \in I$. Such problems are normally solved by a branch-and-bound method that builds a search tree by branching on binary variables. For simplicity of exposition, we assume no cutting planes are added to the constraint set during the search. Let z^* be the optimal value of z obtained for (49).

Inference-based postoptimality analysis is based on dual solutions obtained at nodes of the search tree. An LP relaxation of (49) is solved at each node of the tree, namely

$$\min \left\{ z_{\text{LP}} = \boldsymbol{c}^\top \boldsymbol{x} \;\middle|\; A\boldsymbol{x} \geq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0},\ D\boldsymbol{x} \geq \boldsymbol{d} \right\} \tag{50}$$

where $D\boldsymbol{x} \geq \boldsymbol{d}$ consists of branching constraints of the form $x_i \leq 0$ (when branching to $x_i = 0$) or $x_i \geq 1$ (when branching to $x_i = 1$) that are added along the path from the root of the tree to the current node. If one or more of the 0–1 variables has a fractional value in the LP solution, the search creates two branches by setting $x_i = 0$ and $x_i = 1$ for one such variable x_i.

The LP dual of (50) is

$$\max \left\{ \boldsymbol{u}^\top \boldsymbol{b} + \boldsymbol{v}^\top \boldsymbol{d} \;\middle|\; \boldsymbol{u}^\top A + \boldsymbol{v}^\top D \leq \boldsymbol{c}^\top,\ \boldsymbol{u} \geq \boldsymbol{0},\ \boldsymbol{v} \geq \boldsymbol{0} \right\}$$

Let $\boldsymbol{x}_B$ be the variables fixed by $B\boldsymbol{x} \geq \boldsymbol{b}$, and $\bar{\boldsymbol{x}}$ the values to which they are fixed. Then at each leaf node of the tree, one of three cases obtains:

(a) Problem (50) is infeasible. Then there is a dual feasible extreme ray $(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}})$ for which $\boldsymbol{x}_B = \bar{\boldsymbol{x}}_B$ violates the surrogate inequality $\bar{\boldsymbol{u}}^\top A\boldsymbol{x} \geq \bar{\boldsymbol{u}}^\top \boldsymbol{b}$.

(b) Problem (50) has an optimal solution $\bar{\boldsymbol{x}}$ with value $\bar{z}_{\text{LP}}$ in which $\bar{x}_j \in \{0,1\}$ for $j \in J$, which means that $\bar{\boldsymbol{x}}$ is feasible in (49) and therefore a candidate solution. Then if $(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}})$ is an optimal dual solution, $\boldsymbol{x}_B = \bar{\boldsymbol{x}}_B$ violates the surrogate inequality $\bar{\boldsymbol{u}}^\top A\boldsymbol{x} - \boldsymbol{c}^\top \boldsymbol{x} > \bar{\boldsymbol{u}}^\top \boldsymbol{b} - \bar{z}_{\text{LP}}$.

(c) Problem (50) has an optimal solution with value $\bar{z}_{\text{LP}}$ that is no smaller than the value $\bar{z}_{\min}$ of the best candidate solution found so far. Then if $(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}})$ is an optimal dual solution, $\boldsymbol{x}_B = \bar{\boldsymbol{x}}_B$ violates the surrogate inequality $\bar{\boldsymbol{u}}^\top A\boldsymbol{x} - \boldsymbol{c}^\top \boldsymbol{x} > \bar{\boldsymbol{u}}^\top \boldsymbol{b} - \bar{z}_{\min}$.

To conduct postoptimality analysis, we first observe that if a constraint i has a vanishing dual multiplier $\bar{u}_i$ at every leaf node, then the constraint plays no role in the proof of optimality (or infeasibility, but for simplicity we suppose that the problem is feasible). Constraint i can therefore be dropped without changing the optimal solution.

Beyond this, we can study which perturbations in the problem data allow one to prove an optimal value of at least z^*. We first observe that setting $\boldsymbol{x_B} = \bar{\boldsymbol{x}}_{\boldsymbol{B}}$ at a leaf node falsifies a logical clause that negates these settings. For example, if a leaf node is reached by setting $(x_1, x_2, x_3) = (1, 0, 1)$, the clause $\neg x_1 \vee x_2 \vee \neg x_3$ is falsified. We will call this the branching clause at the node. Since the branching search is exhaustive, the set of branching clauses for all the leaf nodes is unsatisfiable. We also observe that since $\boldsymbol{x_B} = \bar{\boldsymbol{x}}_{\boldsymbol{B}}$ violates the surrogate inequality at any given leaf node, the surrogate inequality must imply a logical clause that implies the branching clause at that node. We will call this the surrogate clause. To continue the example, if the surrogate inequality at the leaf node is $-4x_1 + 3x_2 - 2x_3 \geq -3$, it implies the surrogate clause $\neg x_1 \vee x_2$, which implies the branching clause $\neg x_1 \vee x_2 \vee \neg x_3$. We conclude that after a perturbation, the original search tree remains exhaustive and proves an optimal value of at least z^* if the perturbed surrogate inequality still implies the original surrogate clause at each node. The actual optimal value of the perturbed problem may, of course, be larger than z^*.

This analysis leads to a system of inequalities whose satisfaction by a perturbation ensures that the optimal value will not fall below z^*. For present purposes, it suffices to illustrate how this works in the many-valued logic example presented in Section 4.2. Recall that the objective is to find the smallest truth value t' for which we can infer $\boxed{\geq t'}((P \supset Q) \supset \neg P)$, given $\boxed{\leq p}\, P$ and $\boxed{\leq q}\, Q$. For the sake of illiustration, we suppose $(p, q) = (0.3, 0.7)$. The smallest t' is found by minimizing t' subject to the MILP constraint set in (33), which is reproduced below in a consistent format (omitting nonnegativity constraints on the variables):

$$
\begin{array}{lll}
1.\ \ t' - \delta' \geq 0 & (u_1) & 6.\ \ t' + \delta' + t + v(P) \geq 2 \quad (u_6) \\
2.\ \ -t + v(P) - \delta' \geq -2 & (u_2) & 7.\ \ -t \geq -1 \quad (u_7) \\
3.\ \ t - \delta \geq 0 & (u_3) & 8.\ \ t + v(P) - v(Q) + \delta \geq 1 \quad (u_8) \\
4.\ \ -v(P) + v(Q) - \delta \geq -1 & (u_4) & 9.\ \ -v(P) \geq -0.3 \quad (u_9) \\
5.\ \ -v(Q) \geq -0.7 & (u_5) & \delta, \delta' \in \{0, 1\}
\end{array}
\tag{51}
$$

The dual multipliers u_i are indicated to the right of the constraints. The search tree for this problem consists of of three nodes, as shown in Fig. 6. Since δ' is a fraction 0.35 in the solution of the LP relaxation at the root node, the search branches on δ' to create nodes 2 and 3. The LP relaxation at node 2 has an integral solution with $(\delta, \delta') = (0, 0)$ and optinal value $\bar{t}'_{\text{LP}} = 0.7$, so that case (b) applies. The solution at node 3 is nonintegral, but since the optimal value $\bar{t}'_{\text{LP}} = 1$ is worse than the best previous solution value 0.7, case (c) applies, and there is no need for further branching. The minimum value of t' is therefore 0.7, and so $(P \supset Q) \supset \neg P$ has a truth value no less than 0.7.

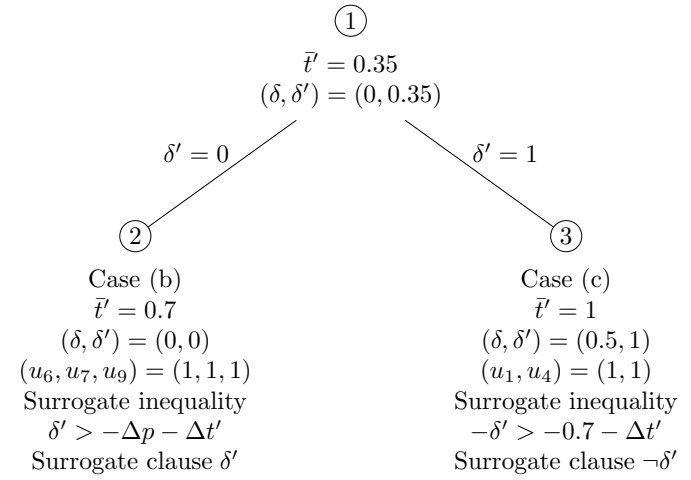

Fig. 6 Branch-and-bound search tree for the multi-valued logic problem (33). Only nonzero dual multipliers u_i are indicated. The surrogate inequalities reflect a perturbation of the problem data.

We are particularly interested in the sensitivity of the solution to changes in truth value bounds $(p, q) = (0.3, 0.7)$, which are encoded in constraints 5 and 9. Figure 6 indicates the nonzero dual multipliers at each leaf node, and we note that the multiplier u_5 corresponding to $v(Q) \leq 0.7$ is zero at both leaf nodes. This means that the lower bound of 0.7 on the truth value of $(P \supset Q) \supset \neg P$ remains valid even if the bound on $v(Q)$ is dropped.[6]

To proceed to a deeper analysis, we begin by examining node 2. To obtain the surrogate inequality, we first take a linear combination of perturbed constraints 6, 7, and 9 with multipliers $(u_6, u_7, u_9) = (1, 1, 1)$. Constraint 9 is perturbed by replacing the bound 0.3 with $0.3 + \Delta p$ to allow us to account for the effect of perturbations in this bound. Constraint 6 is perturbed by replacing t' with $t' + \Delta t'$ to permit an investigation of which lower bounds for t' other than 0.7 can be proved after the perturbation. Since case (b) applies to node 2, we get the surrogate inequality $\delta' > -\Delta p - \Delta t'$. This implies the single-literal clause δ' when $\Delta p = \Delta t' = 0$, which means that δ' is the surrogate clause. The surrogate inequality continues to imply clause δ' as long as

$$\Delta p + \Delta t' \leq 0 \tag{52}$$

Node 3, which corresponds to case (c), similarly yields the surrogate inequality $-\delta' > -0.7 - \Delta t'$. This implies the clause $\neg\delta'$ when $\Delta t' = 0$, which is therefore the surrogate clause. The surrogate inequality continues to imply $\neg\delta'$ as long as

$$\Delta t' \leq 0.3 \tag{53}$$

[6] This does not mean that the bound on $v(Q)$ cannot affect the actual minimum value of t'. For example, setting $v(Q) = 0.1$ results in $\bar{t}' = 0.9$. Yet $\bar{t}'$ is still at least 0.7.

The lower bound 0.7 remains proven for t' as long as the perturbation Δp satisfies (52) and (53) with $\Delta t' = 0$; that is, as long as $\Delta p \leq 0$, which is to say that p (currently 0.3) can be reduced but not increased.

If we wish to know under what conditions a lower bound of 0.9 (rather than 0.7) can be proved for t', we can let $\Delta t' = 0.2$ in (52) and (53). Since $\Delta t' = 0.2$ satisfies (53), condition (52) tells us that $\Delta p \leq -0.2$ suffices to prove $t' \geq 0.7 + 0.2$. Thus if the upper bound on $v(P)$ is reduced from 0.3 to to 0.1, a lower bound of 0.9 can be proved for the truth value of $(P \supset Q) \supset \neg P$, regardless of the bound (if any) given for $v(Q)$. As it happens, this is exactly the optimal value of t' after the perturbation.

The upshot of this analysis is that $(P \supset Q) \supset \neg P$ has a truth value of at least 0.7, given that P's truth value is at most 0.3, regardless of the upper bound we place on Q's truth value. The guaranteed truth value 0.7 remains valid so long as the upper bound 0.3 on P's truth value is not relaxed. In fact, if it is tightened (reduced), the guaranteed truth value of $(P \supset Q) \supset \neg P$ increases by an equal amount, but if it is relaxed, the guaranteed truth value is reduced by an equal amount.

The practicality of this type of comprehensive analysis depends on the number of leaf nodes that generate relevant surrogate inequalities. The total of number of leaf nodes can be very large, but it is likely that only a fraction of them generate surrogate inequalities that contain perturbation terms for the constraints of interest. The remainder of the leaf nodes can be ignored. If we are interested only in learning which constraints play a role in deducing the optimal solution, we need only observe which constraints have a positive dual multiplier in at least one leaf node. There is no need to generate and store surrogate inequalities. This limited form of analysis can be performed on any problem instance that can be solved by a branch-and-bound method.

7.3 MILP Postoptimality Analysis with Decision Diagrams

A second method of postoptimality analysis for MILP, described in [80], generates a BDD that represents near-optimal solutions of the problem. Such a BDD is convenient for conducting several types of postoptimality analysis. One can efficiently build a BDD that compactly stores all such solutions during the normal branch-and-bound search for an optimal solution. A simple procedure ("sound reduction") allows it to be further compressed, often by several orders of magnitude, by introducing certain infeasible solutions that do not affect postoptimality analysis. As a result, very large optimization problems can be analyzed in this fashion.

We illustrate the idea by applying it to the problem of inference in propositional logic. Consider again the set $F = \{f_1, f_2, f_3\}$ of formulas listed in (43). Suppose we wish to determine the minimum number of atomic propositions x_i that can be false if F is to be satisfied. This can be ascertained by minimizing $\sum_i (1 - x_i)$ subject to F, a problem that can be solved by integer programming. As it happens, at least two variables must be false. We would like to know which of the formulas in F play a role in this deduction.

This question can be addressed by introducing 0–1 variables y_1, y_2, y_3 that respectively indicate whether f_1, f_2, f_3 are enforced. We augment F by adding formulas $y_i \supset f_i$ for $i = 1, 2, 3$ to obtain F'. The minimum of $\sum_i (1 - x_i)$ subject to F' is of course zero, since the y_is can all be 0, in which case no f_i is enforced. However, we can investigate the role of the f_is by building a BDD that represents certain *near-optimal solutions* of this problem; namely, solutions with value 0, 1, or 2. Such a BDD appears in Fig. 7(a).

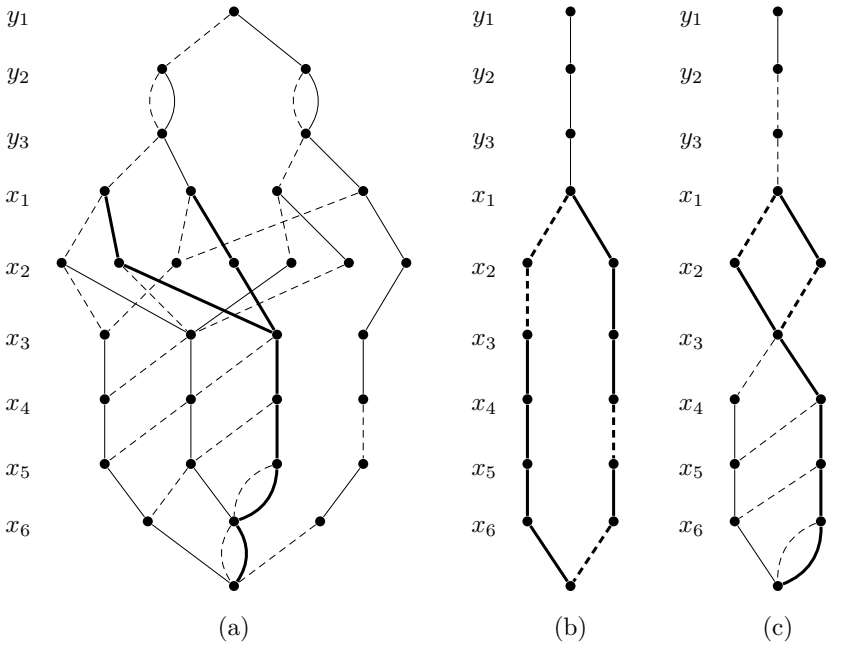

Fig. 7 (a) BDD for postoptimality analysis of inference from formulas (43). (b) BDD after fixing $(y_1, y_2, y_3) = (1, 1, 1)$. (c) BDD after fixing $(y_1, y_2, y_3) = (1, 0)$. Thick arcs indicate optinal values of $\boldsymbol{x}$.

An optimal solution subject to F' can be found by solving an easy shortest path problem in the BDD of Fig. 7(a). Since we are minimizing the number of false x_is, we place a cost (length) of 1 on dashed arcs in Fig, 7 that correspond to values of the x_is, and view all other arcs as having cost 0. The shortest top-to-bottom path has zero cost, indicating that an optimal solution has value 0, as expected. Now we can determine the result of enforcing any subset of f_is by fixing the corresponding y_is to 1 and computing shortest paths in the simplified BDD that results.

We can see right away that f_2 has no bearing on the problem, since the value of y_2 has no effect on any path length. If we enforce all the f_is by fixing all y_is to 1, we get the BDD in Fig. 7(b), in which the shortest path length is 2, as expected. If we enforce only f_1, we obtain the BDD of Fig. 7(c), in which the shortest path has

length 1. Thus without premise f_3, the minimum number of false atoms drops to 1. It is similarly shown that without f_1, the number drops to 0.

The BDDs also allow inferences regarding possible values of the variables x_i. If all three formulas are enforced, as in Fig. 7(b), we see immediately that the number of false atoms can be reduced to 2 only by setting x_3 and x_5 to true. If we enforce only f_1, as in Fig. 7(c), we see that the number of false atoms can be reduced to 1 only by setting x_1 or x_2 to false.

References

1. Akers, S.B.: Binary decision diagrams. IEEE Transactions on Computers **C-27**, 509–516 (1978)
2. Amano, K., Maruoka, A.: On learning monotone boolean functions under the uniform distribution. Theoretical Computer Science **350**, 3–12 (2006)
3. Andersen, K.A., Hooker, J.N.: Bayesian logic. Decision Support Systems **11**, 191–210 (1994)
4. Andersen, K.A., Hooker, J.N.: Determining lower and upper bounds on probabilities of atomic propositions in sets of logical formulas represented by digraphs. Annals of Operations Research **65**, 1–20 (1996)
5. Andersen, K.A., Hooker, J.N.: A linear programming framework for logics of uncertainty. Decision Support Systems **16**, 39–53 (1996)
6. Andersen, K.A., Pretolani, D.: Easy cases of probabilistic satisfiability. Annals of Mathematics and Artificial Intelligence **33**, 69–91 (2001)
7. Anderson, J.R., Bothell, D., Byrne, M.D., Douglass, S.A., Lebiere, C., Qin, Y.: An integrated theory of the mind. Psychological Review **111**, 1036–1060 (2004)
8. Anthony, M.: Probabilistic learning and Boolean functions. In: Y. Crama, P.L. Hammer (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, *Encyclopedia of Mathematics and its Applications*, vol. 134, pp. 197–220. Cambridge University Press, Cambridge (2010)
9. Bell, C., Nerode, A., Ng, R.T., Subrahmanian, V.S.: Mixed integer progamming methods for computing nonmonotonic deductive databases. Journal of the ACM **41**, 1178–1215 (1994)
10. Ben-Eliyahu–Zohary, R.: An incremental algorithm for generating all minimal models. Artificial Intelligence **169**, 1–22 (2005)
11. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik **4**, 238–252 (1962). DOI 10.1007/BF01386316
12. Bergman, D., Cire, A.A., van Hoeve, W.J., Hooker, J.N.: Decision Diagrams for Optimization. Springer (2016)
13. Blum, A., Burcht, C., Langford, J.: On learning monotone Boolean functions. In: Proceedings 39th Annual Symposium on Foundations of Computer Science, pp. 408–415 (1998)
14. Bollig, B., Sauerhoff, M., Sieling, D., Wegener, I.: Binary decision diagrams. In: Y. Crama, P.L. Hammer (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, pp. 473–505. Cambridge University Press, Cambridge, UK (2010)
15. Boole, G.: An Investigation of the Laws of Thought, On Which are Founded the Mathematical Theories of Logic and Probabilities. Walton and Maberly, London (1854)
16. Borkar, V.S., Chandru, V., Mitter, S.K.: Mathematical programming embeddings of logic. Journal of Automated Reasoning **29**, 91–106 (2002)
17. Boros, A., Hammer, P.L., Hooker, J.N.: Predicting cause-effect relationships from incomplete discrete observations. SIAM Journal on Discrete Mathematics **7**, 531–543 (1994)
18. Boros, E., Hammer, P.L.: Pseudo-Boolean optimization. Discrete Applied Mathematics **123**, 155–225 (2002)
19. Boros, E., Hammer, P.L., Hooker, J.N.: Boolean regression. Annals of Operations Research **58**, 201–226 (1995)

20. Brun, T.: Structure probabiliste en logique des propositions. Memoire d'ingénieur, École des hautes études commerciales, Montréal (1988)
21. Bryant, R.E.: Symbolic boolean manipulation with ordered binary decision diagrams. ACM Computing Surveys **24**, 293– 318 (1992)
22. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. Journal of Artificial Intelligence Research **70**, 245–317 (2021)
23. Chandru, V., Hooker, J.N.: Extended Horn clauses in propositional logic. Journal of the ACM **38**, 203–221 (1991)
24. Chandru, V., Hooker, J.N.: Optimization Methods for Logical Inference. Wiley (1999)
25. Charnes, A., Cooper, W.W.: Programming with linear fractional functionals. Naval Research Logistics Quarterly **9**, 181–186 (1962)
26. Chen, T., Braga-Neto, U.M.: Statistical detection of boolean regulatory relationships. IEEE/ACM Transactions on Computational Biology and Bioinformatics **10**, 1310–1321 (2017)
27. Chinneck, J.W.: An effective polynomial-time heuristic for the minimum-cardinality IIS set-covering problem. Annals of Mathematics and Artificial Intelligence **17**, 127–144 (1996)
28. Chinneck, J.W.: Finding a useful subset of constraints for analysis in an infeasible linear program. INFORMS Journal on Computing **9**, 111–129 (1997)
29. Chinneck, J.W.: Fast heuristics for the maximum feasible subsystem problem. INFORMS Journal on Computing **13**, 171–256 (2001)
30. Chinneck, J.W., Dravnieks, E.W.: Locating minimal infeasible constraint sets in linear programs. ORSA Journal on Computing **3**, 157–168 (1991)
31. Cozman, F.G., Marinescu, R., Lee, J., Gray, A., Riegel, R., Bhattacharjya, D.: Markov conditions and factorization in logical credal networks. International Journal of Approximate Reasoning **172** (2024)
32. Dawande, M., Hooker, J.N.: Inference-based sensitivity analysis for mixed integer/linear programming. Operations Research **48**, 623–634 (2000)
33. Dempster, A.P.: Upper and lower probabilities induced by a multivaued mapping. Annals of Mathematical Statistics **38**, 325–339 (1967)
34. Dempster, A.P.: A generalization of bayesian inference. Journal of the Royal Statistical Society (Series B) **30**, 205–247 (1968)
35. Devriendt, J., Gleixner, A., Nordström, J.: Learn to relax: Integrating 0–1 integer linear programming with pseudo-Boolean conflict-driven search. Constraints **26**, 26–55 (2021)
36. Devriendt, J., Gocht, S., Demirović, E., Nordström, J., Stuckey, P.: Cutting to the core of pseudo-boolean optimization: Combining core-guided search with cutting planes reasoning. In: National Conference on Artificial Intelligence (AAAI 2021), pp. 3750–3758 (2021)
37. Dugat, V., Sandri, S.: Complexity of hierarchical trees in evidence theory. ORSA Journal on Computing **6**, 37–49 (1994)
38. Elffers, J., Nordström, J.: Divide and conquer: Towards faster pseudo-Boolean solving. In: International Joint Conference on Artificial Intelligence (IJCAI 2018), pp. 1291–1299 (2018)
39. Elffers, J., Nordström, J.: A cardinal improvement to pseudo-Boolean solving. In: National Conference on Artificial Intelligence (AAAI 2020), pp. 1495–1503 (2020)
40. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: R.A. Kowalski, K.A. Bowen (eds.) Proceedings of the 5th International Conference and Symposium on Logic Programming, pp. 1070–1080 (1988)
41. Georgakopolous, G., Kavvadias, D., Papadimitriou, C.H.: Probabilistic satisfiability. Journal of Complexity **4**, 1–11 (1988)
42. Haenni, R., Romeijn, J.W., Wheeler, G., Williamson, J.: Possible semantics for a common framework of probabilistic logics. In: V.N. Huynh, Y. Nakamori, H. Ono, J. Lawry, V. Kreinovich, H.T. Nguyen (eds.) Proceedings of the International Workshop on Interval/Probabilistic Uncertainty and Non Classical Logics, pp. 268–279. Springer (2008)
43. Hähnle, R.: Many-valued logic and mixed integer programming. Annals of Mathematics and Artificial Intelligence **12**, 231–263 (1994)
44. Haibe-Kains, B., Adam, G.A., Hosny, A., Khodakarami, F., Waldron, L., Wang, B., McIntosh, C., Goldenberg, A., Kundaje, A., Greene, C.S., Broderick, T., Hoffman, M.M., Leek, J.T., ,

Korthauer, K., Huber, W., Brazma, A., Pineau, J., Tibshirani, R., Hastie, T., Ioannidis, J.P.A., Quackenbush, J., Aerts, H.J.W.L.: Artificial intelligence and the value of transparency. Nature **586**, E14–E15 (2020)
45. Hailperin, T.: Boole's Logic and Probability, *Studies in Logic and the Foundations of Mathematics*, vol. 85. North-Holland, Amsterdam (1976)
46. Hansen, P., Jaumard, B.: Probabilistic logic. In: J. Kohlas, R. Kruse (eds.) Algorithms for Uncertainty and Defeasible Reasoning, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, vol. 5, pp. 321–368. Springer, Berlin (2000)
47. Hansen, P., Perron, S.: Merging the local and global approaches to probabilistic satisfiability. International Journal of Approximate Reasoning **47**, 125–140 (2008)
48. Hooker, J.N.: A mathematical programming model for probabilistic logic. Tech report 05-88-89, Graduate School of Industrial Administration, Carnegie Mellon University (1988)
49. Hooker, J.N.: Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction. Wiley, New York (2000)
50. Hooker, J.N.: Decision diagrams and dynamic programming. In: C. Gomes, M. Sellmann (eds.) CPAIOR 2013 Proceedings, pp. 94–110. (2013)
51. Hooker, J.N.: Projection, consistency, and George Boole. Constraints **21**, 59–76 (2016)
52. Hooker, J.N.: Logic-Based Benders Decomposition: Theory and Applications. Springer (2024)
53. Hooker, J.N., Ottosson, G.: Logic-based Benders decomposition. Mathematical Programming **96**, 33–60 (2003)
54. Jaumard, B., Hansen, P., Aragão, M.P.: Column generation methods for probabilistic logic. INFORMS Journal on Computing **3**, 135–148 (1991)
55. Jeroslow, R.E., Wang, J.: Solving propositional satisfiability problems. Annals of Mathematics and Artificial Intelligence **1**, 167–188 (1990)
56. Jeroslow, R.G.: Representability in mixed integer programming, I: Characterization results. Discrete Applied Mathematics **17**, 223–243 (1987)
57. Jeroslow, R.G.: Mixed Integer Formulation, *Logic-Based Decision Support*, vol. 40. North-Holland, Amsterdam (1989)
58. Kane, T.B.: Maximum entropy in Nielsson's probabilistic logic. In: Proceedings of IJCAI 1989, pp. 442–447. Morgan Kaufmann (1989)
59. Kavvadias, D., Papadimitriou, C.H.: A linear programming approach to reasoning about probabilities. Annals of Mathematics and Artificial Intelligence **1**, 189–206 (1990)
60. Kim, T.W., Hooker, J.N., Donaldson, T.: Taking principles seriously: A hybrid approach to value alignment in artificial intelligence. Journal of Artificial Intelligence Research **70**, 871–890 (2021)
61. Klinov, P., Parsia, B.: On improving the scalability of checking satisfiability in probabilistic description logics. In: L. Godo, A. Pugliese (eds.) International Conference on Scalable Uncertainty Management, pp. 138–149. Springer, Berlin (2009)
62. Klinov, P., Parsia, B.: A hybrid method for probabilistic satisfiability. In: N. Bjørner, V. Sofronie-Stokkermans (eds.) 23rd International Conference on Automated Deduction (CADE), pp. 354–368. Springer (2011)
63. Lebiere, C., Anderson, J.R.: A connectionist implementation of the ACT-R production system. In: Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society, pp. 635–640 (1993)
64. Liu, G., Janhunen, T., Niemelä, I.: Answer set programming via mixed integer programming. In: Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning, pp. 32–42 (2012)
65. Liu, G., Janhunen, T., Niemelä, I.: Introducing real variables and integer objective functions to answer set programming. In: Proceedings of the International Conference on Applications of Declarative Programming and Knowledge Management, pp. 118–135 (2014)
66. Marek, W., Nerode, A., Remmel, J.B.: The stable models of a predicate logic program. Journal of Logic Programming **21**, 129–154 (1994)
67. Marinescu, R., Lee, J., Bhattacharjya, D., Cozman, F., Gray, A.: Abductive reasoning in logical credal networks. In: 38th Conference on Neural Information Processing Systems (NeurIPS), pp. 635–640 (2024)

68. Marinescu, R., Qian, H., Gray, A., Bhattacharjya, D., Barahona, F., Gao, T., Riegel, R., Sahu, P.: Logical credal networks. In: 36th Conference on Neural Information Processing Systems (NeurIPS), pp. 635–640 (2022)
69. Mauá, D.D., Cozman, F.G.: Thirty years of credal networks: Specification, algorithms and complexity. International Journal of Approximate Reasoning **126**, 133–157 (2020)
70. Mukherjee, S., Pelech, S., Neve, R.M., Kuo, W.L., Ziyad, S., Spellman, P.T., Gray, J.W., Speed, T.P.: Sparse combinatorial inference with an application in cancer biology. Bioinformatics **25**, 265–271 (2009)
71. Mukherjee, S., Speed, T.P.: Sparse, noisy Boolean functions. Technical report, University of California, Berkeley (2007)
72. Newell, A.: Unified Theories of Cognition. Harvard University Press, Cambridge, MA (1994)
73. Newell, A., Shaw, J.C., Simon, H.A.: Report on a general problem-solving program. In: Proceedings of the International Conference on Information Processing (IFIP), vol. 1, pp. 256–264. UNESCO, Paris (1959)
74. Newell, A., Simon, H.A.: The logic theory machine: A complex information processing system. Tech report P-868, RAND Corporation (1956)
75. Nilsson, N.J.: Probabilistic logic. Artificial Intelligence **28**, 71–87 (1986)
76. Nilsson, N.J.: Probabilistic logic revisited. Artificial Intelligence **59**, 39–42 (1993)
77. Osorio, M., Díaz, J., Santoyo, A.: 0-1 integer programming for computing semi-stable semantics of argumentation frameworks. Computación y Sistemas **21** (2017)
78. Popov, M., Balyo, T., Iser, M., Ostertag, T.: Construction of decision diagrams for product configuration. In: Proceedings of the 25th International Workshop on Configuration, pp. 108–117 (2023)
79. Sanchez, S.N., Triantaphyllou, E., Chen, J., Liao, T.W.: An incremental learning algorithm for constructing boolean functions from positive and negative examples. Computers and Operations Research **29**, 1677–1700 (2002)
80. Serra, T., Hooker, J.N.: Compact representation of near-optimal integer programming solutions. Mathematical Programming **182**, 199–232 (2020)
81. Shafer, G.: A Mathematical Theory of Evidence. Princeton University Press, Princeton, NJ (1976)
82. Sloan, R.H., Szörényi, B., Turán, G.: Learning boolean functions with queries. In: Y. Crama, P.L. Hammer (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, *Encyclopedia of Mathematics and its Applications*, vol. 134, pp. 221–256. Cambridge University Press, Cambridge (2010)
83. Smirnov, P., Berg, J., Järvisalo, M.M.: Pseudo-Boolean optimization by implicit hitting sets. In: L.D. Michel (ed.) Principles and Practice of Constraint Programming (CP 2022), pp. 51:1–20. Springer (2022)
84. SolverMax: Solver performance: 1989 vs 2024. 5 January (2024). URL www.solvermax.com /blog/solver-performance-1989-vs-2024
85. Walmsley, J.: Artificial intelligence and the value of transparency. AI and Society **36**, 585–595 (2021)
86. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. Society for Industrial and Applied Mathematics, Philadelphia (2000)
87. Wegener, I.: BDDs—design, analysis, complexity, and applications. Discrete Applied Mathematics **138**, 229–251 (2004)
88. Wilson, N.: Algorithms for Dempster-Shafer theory. In: J. Kohlas, R. Kruse (eds.) Algorithms for Uncertainty and Defeasible Reasoning, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, vol. 5, pp. 421–476. Springer, Berlin (2000)
89. Zhou, J., Chen, F. (eds.): Human and Machine Learning: Visible, Explainable, Trustworthy, and Transparent. Springer, Dordrecht (2018)