Constraints (2016) 21:59–76
DOI 10.1007/s10601-015-9201-2

Projection, consistency, and George Boole

J. N. Hooker[1]

Published online: 18 August 2015

Abstract Although best known for his work in symbolic logic, George Boole made seminal contributions in the logic of probabilities. He solved the probabilistic inference problem with a projection method, leading to the insight that inference (as well as optimization) is essentially a projection problem. This unifying perspective has applications in constraint programming, because consistency maintenance is likewise a form of inference that can be conceived as projection. Viewing consistency in this light suggests a concept of J-consistency, which is achieved by projection onto a subset J of variables. We show how this projection problem can be solved for the satisfiability problem by logic-based Benders decomposition. We also solve it for `among`, `sequence`, `regular`, and `all-different` constraints. Maintaining J-consistency for global constraints can be more effective than maintaining traditional domain and bounds consistency when propagating through a richer structure than a domain store, such as a relaxed decision diagram. This paper is written in recognition of Boole's 200th birthday.

Keywords Projection · Consistency · Optimization · Inference · Satisfiability · Logic-based Benders decomposition · Boole

1 Introduction

Although George Boole is best known for his work in symbolic logic, he made a strikingly original contribution to the logic of probabilities [12, 13]. He formulated the probabilistic inference problem as an optimization problem we now call linear programming. He solved the problem with a projection method we now call Fourier-Motzkin elimination. In a single stroke, he linked the concepts of projection, optimization, and logical inference.

✉ J. N. Hooker
jh38@andrew.cmu.edu

[1] Carnegie Mellon University, Pittsburgh, PA, USA

Boole's insight extends to constraint programming (CP) as well, because consistency maintenance is a form of projection. Projection is, in fact, the root idea that unites the other three concepts. Optimization is projection of the feasible set onto a variable that represents the objective value. Inference, as we will see, can be understood as projection onto a desired subset of variables. Consistency maintenance is a form of inference that is likewise equivalent to projection.

We suggest that this unifying vision can be exploited in CP by addressing consistency maintenance explicitly as a projection problem. Existing types of consistency are already forms of projection, but viewing them in this light suggests a particularly simple type of consistency that has apparently not seen application. We call it J-consistency, which is achieved by projecting the problem's solution set onto a subset J of variables.

Our goal is not to solve a particular problem, but to show how projection is a unifying concept, and to take initial steps in a research program that applies this insight in inference and CP. We first review how Boole united optimization, inference, and projection in his treatment of probability logic. We then turn to propositional logic and note that the resolution procedure, which is closely related to Fourier-Motzkin elimination, achieves J-consistency. Because resolution is generally impractical, we propose a more efficient projection method based on the fact that logic-based Benders decomposition computes a projection. We illustrate how conflict clauses generated during solution of the satisfiability problem can deliver the desired projection.

We next discuss the relationship between consistency and projection. We observe that domain consistency is a particularly simple form of projection, while k-consistency is a less obvious form. We then indicate how achieving J-consistency, which is quite different from k-consistency, can reduce backtracking when the solver propagates through a richer structure than a domain store. One such structure is a relaxed decision diagram, which recent research suggests can be a more effective propagation medium than variable domains [7–9, 14, 15].

We next investigate the projection problem for a few popular global constraints. We find that projections are easy to compute for `among` constraints, relatively easy for `sequence` constraints, and quite straightforward for `regular` constraints. Projection is complicated in principle for `all-different` constraints, but it tends to simplify when the domains are small, making it feasible to project out some of the variables. These results suggest that achieving J-consistency could have practical application in a solver.

In a concluding section, we propose that a natural generalization of bounds consistency that parallels J-consistency is projection of the convex hull of the solution set. This might be called *continuous J-consistency*. Computing this type of consistency is closely related to the project of identifying valid cutting planes, which has long been pursued in mathematical programming.

2 Probability logic

Our presentation of Boole's probability logic largely follows the interpretation of Hailperin [20]. We are given a set $S = \{C_i \mid i \in I\}$ of logical clauses, where each clause C_i has probability π_i of being true. The problem is to deduce the probability of a given clause C_0 ($0 \notin I$).

Boole formulated this inference problem as what we now call a linear programming (LP) problem. Let $x = (x_1, \ldots, x_n)$ be the atomic propositions that appear in S, and let p_v be the (unknown) probability that $(x_1, \ldots, x_n)$ have truth values $v = (v_1, \ldots, v_n)$. If V_i is the

set of truth value assigments v that make C_i true, then $\pi_i = \sum_{v \in V_i} p_v$. The possible range of probabilities π_0 of C_0 is obtained by minimizing and maximizing π_0 subject to these equations and the facts that the probabilities p_v must be nonnegative and sum to one:

$$\left\{ \min / \max \ \pi_0 \;\middle|\; \pi_0 = \sum_{v \in V_0} p_v;\ \sum_{v \in V_i} p_v = \pi_i,\ i \in I;\ \sum_{v \in V} p_v = 1;\ p \geq 0 \right\} \tag{1}$$

This is an LP problem in variables π_0 and p_v for $v \in V$, where V is the set of all 2^n possible truth value assignments to x.

For example, suppose we are given clause set $S = \{x_1, \bar{x}_1 \vee x_2, \bar{x}_2 \vee x_3\}$ in which the three clauses have probabilities 0.9, 0.8, and 0.4, respectively. If we wish to determine a range of probabilities for x_3, the LP problem (1) becomes

$$\begin{gathered} \min / \max \ \pi_0 \\ \begin{bmatrix} 01010101 \\ 00001111 \\ 11110011 \\ 11011101 \\ 11111111 \end{bmatrix} \begin{bmatrix} p_{000} \\ p_{001} \\ p_{010} \\ \vdots \\ p_{111} \end{bmatrix} = \begin{bmatrix} \pi_0 \\ 0.9 \\ 0.8 \\ 0.4 \\ 1 \end{bmatrix} \\ p_{000}, p_{001}, p_{010}, \ldots, p_{111} \geq 0 \end{gathered}$$

The minimum and maximum values of π_0 are 0.1 and 0.4, indicating that the probability of x_3 must lie in this range.

Interestingly, the model (1) was reinvented in the AI community in the 1980s [42], more than a century after an equivalent formulation appeared in Boole's work. The number of variables in the model grows exponentially with the number of atomic propositions, but column generation methods can often overcome this difficulty in practice by allowing the solver to consider only a tiny fraction of the variables [21, 25, 33–36].

Boole solved (1) by a method we now call Fourier-Motzkin elimination [16, 41]. Given a general LP problem that minimizes or maximizes y_0 subject to $y_0 = ay$ and $Ay \geq b$, where $y = (y_1, \ldots, y_n)$, we can compute the projection onto $y_0, \ldots, y_k$ by eliminating variables $y_n, y_{n-1}, \ldots, y_{k+1}$ one at a time. Let S initially be the set of inequalities $ay \geq y_0$, $ay \leq y_0$, and $Ay \geq b$. Each variable y_j is eliminated as follows. For each pair of inequalities in S that have the form $c\bar{y} + c_0 y_j \geq \gamma$ and $d\bar{y} - d_0 y_j \geq \delta$, where $c_0, d_0 > 0$ and $\bar{y} = (y_1, \ldots, y_{j-1})$, we have

$$-\frac{c}{c_0}\bar{y} + \frac{\gamma}{c_0} \leq y_j \leq \frac{d}{d_0}\bar{y} - \frac{\delta}{d_0}$$

or $L \leq y_j \leq U$ for short. We therefore add inequality $L \leq U$ to S for each such pair. This done, we remove from S all inequalities that contain y_j. The inequalities in S at the end of the procedure describe the projection. Eliminating $y_n, \ldots, y_1$ leaves bounds $\ell \leq y_0 \leq u$, which tell us that ℓ and u are the minimum and maximum values of y_0.

The method is generally impractical because S tends to explode unless special structure prevents it. However, by proposing a projection algorithm to solve the probabilistic inference problem, Boole revealed the connections among projection, optimization, and inference.

3 Inference

Inference can be understood as the process of extracting information that relates to a particular question or topic. For example, if S is a constraint set that describes the operation of a factory, we may wish to deduce facts about a certain product P. Let's suppose the constraints in S collectively contain variables $x_1, \ldots, x_n$, and that $x_1, \ldots, x_k$ are relevant to product P. For example, x_1 may be the model of P produced, x_2 the output level of P, x_2 its unit manufacturing cost, and so forth up to x_k. Then we wish to deduce from S all constraints containing $x_1, \ldots, x_k$. We will see that this is a projection problem.

3.1 Inference as projection

To make the connection between inference and projection more precise, we standardize terminology as follows. For $J \subseteq \{1, \ldots, n\}$, let x_J be the tuple of variables in $\{x_j \mid j \in J\}$ arranged in increasing order of indices, and similarly for v_J. Let D_j be the domain of x_j, with $D = D_1 \times \cdots \times D_n$ and $D_J = \prod_{j \in J} D_j$. Projection can be defined semantically by saying that a set $V' \subseteq D_J$ of tuples is the *projection onto* x_J of $V \subseteq D$ when $V' = \{v_J \mid v \in V\}$. This can be written $V' = V|_J$. However, we are also interested in a syntactic concept that tells us when a *constraint set* is a projection onto x_J of another constraint set.

To this end, we define a *constraint* to be an object that *contains* certain variables and is either *satisfied* or *violated* by any given assignment of values to those variables. An assignment can satisfy or violate a contraint only when it fixes all variables in the constraint.[1]

Let $D_J(S)$ be the set of all $v \in D_J$ for which $x_J = v$ satifies S (i.e., satisfies all the constraints in S), where in particular $D(S) = D_{\{1,\ldots,n\}}(S)$. We say that S is a *constraint set over* x when it contains only variables in $x = (x_1, \ldots, x_n)$, perhaps not all. If S is a constraint set over x, then S *implies* constraint C if an assignment to x satisfies C whenever it satisfies S, or $D(S) \subseteq D(\{C\})$.

Let S' and S be constraint sets over x_J and x, respectively. We define S' to be a *projection onto* x_J of S when S' describes the projection onto x_J of S's satisfaction set, or more precisely, $D_J(S') = D(S)|_J$. It is easy to show that projection captures exactly what S implies about about x_J, in the following sense:

Lemma 1 *Let S and S' be constraint sets over x and x_J, respectively. Then set S' is a projection of S onto x_J if and only if S' implies all and only constraints over x_J that are implied by S.*

As an example, consider a constraint set S consisting of the logical clauses in Table 1. The clause set $S' = \{x_1 \vee x_2, x_1 \vee x_3\}$ is a projection of S onto (x_1, x_2, x_3). This means that any clause over (x_1, x_2, x_3) implied by S is implied by S'. The two clauses in S' capture all that can be inferred in terms of atoms x_1, x_2, x_3.

[1] One can, of course, sometimes *infer* from a partial assignment that a constraint must be satisfied or violated. For example, if x_1, x_2 are binary variables, then $x_1 = 1$ implies that $x_1 + x_2 \geq 1$ is satisfied. But for purposes of definition, it is convenient to say that the constraint is not actually satisfied or violated until all of its variables are instantiated.

Table 1 A set of logical clauses

$$
\begin{array}{cccccc}
x_1 & & & \vee x_4 \vee x_5 \\
x_1 & & & \vee x_4 \vee \bar{x}_5 \\
x_1 & & & & \vee x_5 \vee x_6 \\
x_1 & & & & \vee x_5 \vee \bar{x}_6 \\
 & x_2 & & & \vee \bar{x}_5 \vee x_6 \\
 & x_2 & & & \vee \bar{x}_5 \vee \bar{x}_6 \\
 & & x_3 \vee \bar{x}_4 \vee x_5 \\
 & & x_3 \vee \bar{x}_4 \vee \bar{x}_5
\end{array}
$$

3.2 Inference in propositional logic

An elimination procedure based on resolution solves the projection problem for logical clauses. The procedure is the same as Fourier-Motzkin elimination, except that when eliminating variable x_j, it considers pairs of clauses of the form $C \vee x_j$ and $D \vee \bar{x}_j$, where no one variable occurs negated in C and posited in D (or vice-versa). Each pair generates a *resolvent* on x_j, namely $C \vee D$. For example, resolving $x_1 \vee x_2 \vee x_4$ and $x_2 \vee \bar{x}_3 \vee \bar{x}_4$ on x_4 yields $x_1 \vee x_2 \vee \bar{x}_3$. Resolution can in fact be seen as a form of Fourier-Motzkin elimination plus rounding [47].

The following can be shown [27, 29] by modifying the classical completeness proof for resolution in [44, 45]:

Theorem 1 *Given a clause set S over $x = (x_1, \ldots, x_n)$, eliminating variables x_j for $j \notin J$ by resolution (in any order) yields a projection of S onto x_J.*

A projection obtained in this manner will contain all prime implications of S whose variables are the variables in x_J [45]. A prime implication of S is a clause implied by S that is implied by no other clause implied by S.

Like Fourier-Motzkin elimination, resolution tends to be impractical unless there is special structure, particularly when J is small (so that a large number of variables must be eliminated). However, an alternative procedure can be much more efficient, especially when J is small. It is based on the fact that Benders decomposition [6] can generate a projection of the constraint set onto the variables in the master problem. The classical Benders method applies only to problems with an LP subproblem, but we use logic-based Benders decomposiiton, which is suitable for general constraint solving and optimization [28–31].

We apply Benders decomposition to a clause set S as follows. The master problem (initially empty) consists of Benders cuts in the form of clauses over x_J. Each iteration of the Benders method begins by checking if the master problem is infeasible, in which case the procedure terminates. Otherwise a solution $\bar{x}_J$ of the master problem is obtained. This defines a subproblem $S(\bar{x}_J)$ that is the result of fixing x_J to $\bar{x}_J$ in S. If $S(\bar{x}_J)$ is infeasible, a nogood clause (Benders cut) is generated that excludes $\bar{x}_J$, as well as perhaps other values of x_J for which $S(x_J)$ is infeasible for similar reasons. If $S(\bar{x}_J)$ is feasible, a clause (enumerative Benders cut) is generated that excludes only $\bar{x}_J$. (The cut is "enumerative" in the sense that it allows the Benders algorithm to enumerate distinct feasible solutions.) In either case, the Benders cut is added to the master problem, and the process repeats. At termination, the nogood clauses in the master problem define the projection of S onto x_J [29].

This procedure can be implemented by a single depth-first branching algorithm that generates conflict clauses. Let the variables in x_J be first in the branching order. When unit

propagation detects unsatisfiability at a node of the tree, generate conflict clauses and backtrack (see [4] for a survey of these concepts). Subsequent branches must be consistent with the conflict clauses so far generated. When a feasible node is reached, backtrack to the last variable in x_J. When enumeration is complete, the conflict clauses over x_J define the projection of S onto x_J. Because the search backtracks to level $|J|$ when a satisfying solution is found, the algorithm can be practical when J is not too large.

Suppose, for example, that we wish to project the clause set in Table 1 onto x_J for $J = \{1, 2, 3\}$. A branching tree appears in Fig. 1. Each branch first attempts to set $x_j = \text{F}$ and then $x_j = \text{T}$. Conflict clause $x_1 \vee x_5$ is generated at the leftmost leaf node, which means that setting $x_1 = x_5 = \text{F}$ results in failure, and similarly at the other infeasible leaf nodes. When all clauses in Table 1 are satisfied at a node, the search backtracks to level $|J| = 3$. Upon completion of the search, the set of conflict clauses over (x_1, x_2, x_3) is a projection onto x_J, in this case $\{x_1 \vee x_2, x_1 \vee x_3\}$.

See Algorithm 1 for a more precise description of the procedure. Note that conflict clauses are resolved on the last variable x_j in the clauses when this is possible. In the example, $x_1 \vee x_5$ and $x_2 \vee \bar{x}_5$ are resolved to create conflict clause $x_1 \vee x_2$, and $x_1 \vee x_4$ and $x_3 \vee \bar{x}_4$ are resolved to yield $x_1 \vee x_3$.

The algorithm is presented here as a branching procedure to simplify exposition, but more efficient implementations are possible. For example, the algorithm in [18] uses an efficient SAT algorithm to enumerate all *solutions* of the projection. It may suggest an efficient algorithm that generates a clause set describing the projection.

Theorem 2 *Algorithm 1 generates a projection of S onto x_J.*

Proof We will show that Algorithm 1 implements a Benders method in which the nogood Benders cuts are conflict clauses over x_J. The conflict clauses therefore define the projection of S into x_J.

For any node u of the search tree, let $x_J(u)$ be the assignment to variables in x_J along the path from the root node to u. Whenever an infeasible node u generates a conflict clause $C(u)$ over x_J, we add $C(u)$ to the master problem as a nogood clause. Whenever a feasible solution is found at a node u, we may suppose that a clause $C(u)$ excluding precisely $x_J(u)$ is added to the master problem, because the backtracking mechanism ensures that this value of x_J will not be encountered again.

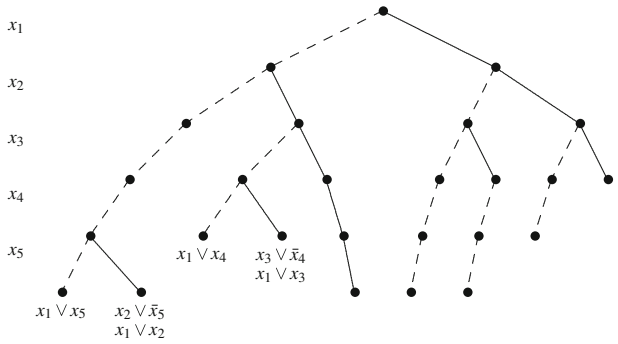

Fig. 1 Branching tree for a SAT instance. *Dashed arcs* indicate $x_j = \text{F}$ and solid arcs $x_j = \text{T}$. Conflict clauses are shown at failure nodes. Solutions are found at remaining leaf nodes, from which the search backtracks to x_3

Algorithm 1 Given a clause set S over $(x_1, \ldots, x_n)$, this recursive algorithm computes a projection of S onto $x_J = (x_1, \ldots, x_k)$. The parameter j of SAT is the current depth in the search tree, and $v = (v_1, \ldots, v_j)$ are the values to which $(x_1, \ldots, x_j)$ have been fixed. The returned value is k if the search is backtracking from a feasible solution to level k, and n otherwise. The global variable N is the set of conflict clauses (nogoods) generated so far. At termination, the clauses in N whose variables are in x_J comprise a projection onto x_J

Let $N = \emptyset$.
SAT(1, 0, S).
If $x_1 = 1$ does not violate N then SAT(1, 1, S).

Function SAT(j, v, S)
  If $v_j = 0$ then let S contain results of unit propagation on $S \cup \{\neg x_j\}$.
  Else let S contain results of unit propagation on $S \cup \{x_j\}$.
  If S contains the empty clause then
    Generate conflict clauses and add them to N.
    Add to S all resolvents on x_j of clauses in S whose last variable is x_j.
    Return(n).
  If S fixes $x_1, \ldots, x_n$ then return(k) (feasible solution found).
  If $(x_J, x_{j+1}) = (v, 0)$ does not violate N then
    Let $\ell = \text{SAT}(j + 1, (v, 0), S)$.
  If $j < \ell$ then
    If $(x_J, x_{j+1}) = (v, 1)$ does not violate N then
      Let $\ell = \text{SAT}(j + 1, (v, 1), S)$.
  Return(ℓ).

If a node u that generates $C(u)$ is at level k or lower, then $x_J(u)$ is a solution of the current master problem, and $C(u)$ a nogood that excludes it. If u is above level k, then some completion down to level k of the assignment $x_J(u)$ is a solution of the master problem, because otherwise resolution of conflict clauses would have produced a conflict clause that excludes u. In either case, u can be regarded as generating a nogood $C(u)$ that excludes a solution of the master problem. A feasible node u gives rise to a feasible solution $\bar{x}$ of S, in which case $\bar{x}_J$ is a solution of the master problem that is excluded by $C(u)$.

Because conflict clauses are resolved, all values of x_J that have no feasible extension are excluded by some conflict clause. This means that the tree search terminates precisely when the master problem no longer has a solution. The search therefore implements a Benders algorithm, and the projection of S onto x_j is defined by the conflict clauses $C(u)$. □

Other adaptations of resolution and Fourier-Motzkin elimination can be used to compute projections for cardinality clauses [24], 0–1 linear inequalities [26], and general integer linear inequalities [48]. Polyhedral projection methods have been studied in the AI community as well [32].

4 Consistency

Popular forms of consistency maintenance already compute projections, but viewing them from this perspective can suggest new and possibly useful forms of consistency.

4.1 Consistency maintenance as projection

Domain consistency is achieved by projecting the solution set onto each individual variable. A constraint set S over x is domain consistent when for each variable x_j and each value $v \in D_j$, the assignment $x_j = v$ is part of some assignment $x = v$ that satisfies S. This is equivalent to saying that $\{x_j \in D_j\}$ is a projection of S onto x_j, or $D_j = D(S)|_{\{j\}}$, for $j = 1, \ldots, n$.

Achieving domain consistency for S at each node of the search tree avoids backtracking, because it allows us to assign any value in D_j to x_j when branching on x_j (if S is satisfiable) without creating an infeasible subtree. Since it is generally impractical to achieve domain consistency for S as a whole, solvers typically maintain domain consistency (or an approximation of it) for some individual constraints in S and propagate the reduced domains through a domain store. This tends to reduce the search tree due to smaller domains.

Another type of consistency related to backtracking is k-consistency. It is again achieved by projection, but not by projecting the entire constraint set. It projects only subsets of constraints over k variables onto subsets of $k-1$ variables.

A constraint set S over x is k-consistent when for every $J \subseteq \{1, \ldots, n\}$ with $|J| = k-1$, every assignment $x_J = v_J \in D_J$ that does not violate S, and every variable x_j not in x_J, there is an assignment $x_j = v_j \in D_j$ for which $(x_J, x_j) = (v_J, v_j)$ does not violate S. A constraint set S is *strongly* k-consistent when it is ℓ-consistent for $\ell = 1, \ldots, k$.

To relate k-consistency to projection, we define S_J to be the set of constraints in S that are over x_J. Then x_J satisfies S_J if and only if x_J violates no constraints in S. This implies the following:

Lemma 2 *A constraint set S over x is k-consistent if and only if $D_J(S_J) = D_{J\cup\{j\}}(S_{J\cup\{j\}})|_J$ for all $J \subseteq \{1, \ldots, n\}$ with $|J| = k-1$ and all $j \notin J$.*

k-consistency is not an obvious generalization of domain consistency, because 1-consistency is not achieved by projecting the constraint set onto individual variables.

Strong k-consistency avoids backtracking when one branches on $x_1, \ldots, x_n$ if the primal graph of S has width less than k with respect to this ordering [17]. However, it is impractical to maintain strong k-consistency ($k > 1$) for S as a whole. Furthermore, when propagation is through a domain store as in standard solvers, there is no point in maintaining k-consistency rather than domain consistency for individual constraints.

4.2 J-Consistency

We propose a type of consistency that is more directly related to projection and naturally generalizes domain consistency. Let S be *J-consistent* when some $S' \subseteq S$ is a projection of S onto x_J. That is, S contains constraints that describe its projection onto x_J, or $D_J(S_J) = D(S)|_J$. If we view S as containing the in-domain constraints $x_j \in D_j$, S is domain consistent if and only if it is $\{j\}$-consistent for $j = 1, \ldots, n$.

Due to Theorem 1, resolution on variables x_j for $j \notin J$ achieves J-consistency for SAT. The Benders procedure described in the previous section also achieves J-consistency, due to Theorem 2.

As in the case of domain consistency, we focus on maintaining J-consistency for individual constraints. If we branch on variables in the order $x_1, \ldots, x_n$, a natural strategy is to project out variables in reverse order $x_n, x_{n-1}, \ldots$ until the computational burden becomes

excessive. We will see below that for some important global constraints, it is relatively easy to project out some or all of the variables.

As in the case of k-consistency, there is no point in maintaining J-consistency for individual constraints when propagation is through a domain store. However, recent research shows that propagation through relaxed decision diagrams can be substantially more effective than domain propagation [7–9, 14, 15]. Maintaining J-consistency could have a significant effect on propagation in this context.

For a simple example of this phenomenon, suppose S consists of the constraints

$$\texttt{among}((x_1, x_2), \{c, d\}, 1, 2) \tag{2}$$

$$(x_1 = c) \Rightarrow (x_2 = d) \tag{3}$$

$$\texttt{alldiff}(x_1, \ldots, x_4) \tag{4}$$

where (2) requires that at least 1 and at most 2 of the variables x_1, x_2 take a value in $\{c, d\}$, and (4) is an `all-different` constraint. The variable domains are $D_1 = D_2 = \{a, b, c, d\}$, $D_3 = \{a, b\}$, and $D_4 = \{c, d\}$. No domain reduction is possible for the individual constraints, and so we must branch on all 4 values of x_1 at the top of the search tree.

However, suppose we propagate through a relaxed decision diagram of width 2 rather than a domain store (which is a decision diagram of width 1). Let's suppose we have already constructed a relaxed decision diagram for constraints (2) and (3), shown in Fig. 2a. The 52 paths from r to t in the diagram represent assignments to $(x_1, \ldots, x_4)$. Thirty-six of these paths represent all of the solutions of (2)–(3), indicating that the diagram in fact represents a relaxation of these constraints.

When branching on x_1, we need only consider values that label outgoing arcs from the root node r of the diagram. Unfortunately, all 4 values appear on these arcs. However, we can reduce branching if we project out x_3, x_4 for the `alldiff` constraint (4). We will see later that the resulting projection is a constraint set consisting of

$$\texttt{alldiff}(x_1, x_2), \quad \texttt{atmost}((x_1, x_2), \{a, b\}, 1), \quad \texttt{atmost}((x_1, x_2), \{c, d\}, 1)$$

where the constraint $\texttt{atmost}(x, V, u)$ is equivalent to $\texttt{among}(x, V, 0, u)$. We can propagate the second `atmost` constraint in a downward pass through the first 2 variables of the relaxed decision diagram of Fig. 2(a). To do so, we let the length of a path be the number of arcs on the path with labels in $\{c, d\}$. Then we indicate, on each arc, the length of the

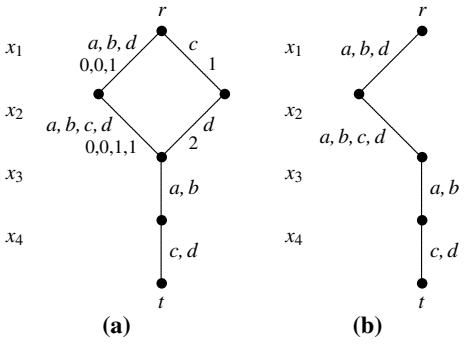

Fig. 2 (**a**) Relaxed decision diagram for an `among` constraint. Arcs with multiple labels represent multiple arcs connecting the same endpoints, each arc with one of the labels. The numbers are part of a mechanism for propagating a projected `alldiff` constraint. (**b**) Decision diagram after propagating the projected `alldiff` constraint

shortest path from r ending with that arc. This allows us to delete any arc assigned a number greater than 1, and any other arcs that lie on no r–t path as a result. We eliminate two arcs, resulting in the smaller decision diagram of Fig. 2b. We therefore need only branch on values a, b, and d. A fuller discussion of propagation through relaxed decision diagrams can be found in [1, 9, 22].

Aside from any advantages in accelerating search, computing projections can infer valuable information from a constraint set. To return to the earlier example of a factory, it may be useful to know what characteritics of a particular product are consistent with the constraint set describing the factory. A projection yields constraints that the product must satisfy. Or if certain variables represent key decisions that are made in the early stages of a project, it may be important to know what options are available for these decisions. Again, projection can answer this question.

4.3 Projection of `among` constraint

Projecting out variables in an `among` constraint [5] is quite simple because each variable elimination yields another `among` constraint. If $x = (x_1, \dots, x_n)$, the constraint $\texttt{among}(x, V, \ell, u)$ requires that at least ℓ and at most u of the variables in x take a value in V. Variable x_n is projected out as follows. Let $\alpha^+ = \max\{\alpha, 0\}$.

Theorem 3 *If $0 \le \ell \le u \le n$, the projection of $\texttt{among}(x, V, \ell, u)$ onto $\bar{x} = (x_1, \dots, x_{n-1})$ is $\texttt{among}(\bar{x}, V, \ell', u')$, where*

$$(\ell', u') = \begin{cases} ((\ell-1)^+, u-1), & \text{if } D_n \subseteq V \quad (a) \\ (\ell, \min\{u, n-1\}), & \text{if } D_n \cap V = \emptyset \quad (b) \\ ((\ell-1)^+, \min\{u, n-1\}), & \text{otherwise} \quad (c) \end{cases}$$

Proof In case (a), x_n must take a value in V, which means that the upper bound u is reduced by 1 and the lower bound ℓ by 1 (unless it is already 0). In case (b), x_n cannot take a value in V, which means that the lower bound ℓ is unchanged. The upper bound u is also unchanged, but if $u = n$ it can be reduced by 1 since there are now only $n-1$ variables. In case (c), x_n can take a value in V, which means that ℓ is modified as in case (a). However, x_n can fail to take a value in V, which means that u is treated as in case (b). □

Variables $x_n, x_{n-1}, \dots, x_1$ are projected out sequentially by applying the theorem recursively. The original constraint is feasible if and only if $\ell' \le u'$ after projecting out all variables.

As an example, let $x = (x_1, \dots, x_5)$, let $V = \{c, d\}$, and let the domains $D_1, \dots, D_5$ be $\{a, b\}$, $\{a, b, c\}$, $\{a, d\}$, $\{c, d\}$, and $\{d\}$. Then while sequentially projecting out $x_5, \dots, x_1$, the original lower bound ℓ becomes $\ell' = (\ell-1)^+, (\ell-2)^+, (\ell-3)^+, (\ell-4)^+$, and $(\ell-4)^+$, while the upper bound u becomes $u' = u-1, u-2, \min\{u-2, 2\}, \min\{u-2, 1\}$, and $\min\{u-2, 0\}$. The constraint is feasible if and only if $(\ell-4)^+ \le \min\{u-2, 0\}$.

4.4 Projection of `sequence` constraint

Fourier-Motzkin elimination provides a fast and convenient method for projecting a `sequence` constraint. The constraint has an integrality property that makes a polyhedral projection technique adequate, and Fourier-Motzkin simplifies to the point that a single generalized `sequence` constraint describes the projection after each variable elimination.

Following standard convention, we assume without loss of generality that the sequence constraint applies to 0-1 variables $x_1, \ldots, x_n$ [23, 46]. It enforces overlapping constraints of the form

$$\texttt{among}((x_{\ell-q+1}, \ldots, x_\ell), \{1\}, L_\ell, U_\ell) \tag{5}$$

for $\ell = q, \ldots, n$, where L_ℓ, U_ℓ are nonnegative integers, and where domain D_j is defined by $\alpha_j \le x_j \le \beta_j$ for $\alpha_j, \beta_j \in \{0, 1\}$. Note that we allow different bounds for different positions ℓ in the sequence. The following theorem provides a recursion for eliminating $x_n, \ldots, x_1$:

Theorem 4 *Given any $k \in \{0, \ldots, n\}$, the projection of the* sequence *constraint defined by (5) onto $(x_1, \ldots, x_k)$ is described by a generalized* sequence *constraint that enforces constraints of the form*

$$\texttt{among}\Big((x_i, \ldots, x_\ell), \{1\}, L^\ell_{\ell-i+1}, U^\ell_{\ell-i+1}\Big) \tag{6}$$

where $i = \ell - q + 1, \ldots, \ell$ for $\ell = q, \ldots, k$ and $i = 1, \ldots, \ell$ for $\ell = 1, \ldots, q-1$. The projection of the sequence *constraint onto $(x_1, \ldots, x_{k-1})$ is given by (6) with $L^\ell_{\ell-i+1}$ replaced by $\hat{L}^\ell_{\ell-i+1}$ and $U^\ell_{\ell-i+1}$ by $\hat{U}^\ell_{\ell-i+1}$, where*

$$\begin{aligned}
\hat{L}^\ell_i &= \begin{cases} \max\{L^\ell_i, L^k_{i+k-\ell} - U^k_{k-\ell}\}, & \text{for } i = 1, \ldots, q-k+\ell, \\ L^\ell_i, & \text{for } i = q-k+\ell+1, \ldots, q \end{cases} \\
\hat{U}^\ell_i &= \begin{cases} \min\{U^\ell_i, U^k_{i+k-\ell} - L^k_{k-\ell}\}, & \text{for } i = 1, \ldots, q-k+\ell, \\ U^\ell_i, & \text{for } i = q-k+\ell+1, \ldots, q \end{cases}
\end{aligned} \tag{7}$$

Proof Constraint (5) is equivalent to

$$L \le \sum_{j=\ell-q+1}^{\ell} x_j \le U \tag{8}$$

and $x_j \in \{0, 1\}$ for $j = \ell - q + 1, \ldots, \ell$, while (6) is equivalent to

$$L^\ell_{\ell-i+1} \le \sum_{j=i}^{\ell} x_j \le U^\ell_{\ell-j+1} \tag{9}$$

and $x_j \in \{0, 1\}$ for $j = i, \ldots, \ell$. As pointed out in [39], the constraint matrix for inequalities (8) has the consecutive ones property. The inequalities therefore describe an integral polyhedron. Imposing 0-1 bounds $L_j \le x_j \le U_j$ has no effect on integrality, so that inequalities (8) and domain bounds describe an integral polyhedron P_n. Because the projection of an integral polyhedron onto any subspace is integral, the projection P_k of P_n onto $(x_1, \ldots, x_k)$ is integral for $k = 1, \ldots, n-1$. The projection of the feasible set of the sequence constraint is therefore described by P_k and $x_j \in \{0, 1\}$ for $j = 1, \ldots, k$.

P_{k-1} can be derived by applying Fourier-Motzkin elimination to P_k. To show that each P_k is described by inequalities of the form (9), note first that P_n is described by inequalities (8) and domain bounds, which correspond to a subset of inequalities (9). Specifically, $(L^\ell_q, U^\ell_q) = (L_\ell, U_\ell)$ for $\ell = q, \ldots, n$ and $(L^\ell_1, U^\ell_1) = (\alpha_\ell, \beta_\ell)$ for $\ell = 1, \ldots, n$. The remaining inequalities (9) can be assumed to be present with nonbinding upper and lower bounds. Now assuming that P_k is described by inequalities of form (9), a tedious but straightforward application of Fourier-Motzkin elimination to (9) shows that inequalities of

precisely the same form describe P_{k-1}. It follows by induction that inequalities of form (9) describe P_k for all k. Furthermore, the resulting bounds in the inequalities describing P_{k-1} are given by (7). This proves the theorem. □

The worst-case complexity of projecting out each variable x_k is $\mathcal{O}(kq)$. We can illustrate the theorem with an example from [29]. Suppose we have a `sequence` constraint that imposes (5) with $n = 6$, $q = 4$, and $L_\ell = U_\ell = 2$ for each ℓ. Variables x_1, x_3, x_4, and x_6 have domain $\{0, 1\}$, while x_2 and x_5 have domain $\{1\}$. There is one feasible solution, namely $x = (1, 1, 0, 0, 1, 1)$. Initially the bounds in (6) are $L_4^\ell = U_4^\ell = 2$ for $\ell = 4, 5, 6$, $(L_1^\ell, U_1^\ell) = (0, 1)$ for $\ell = 1, 3, 4, 6$, and $(L_1^\ell, U_1^\ell) = (1, 1)$ for $\ell = 2, 5$, with all other $L_i^\ell = -\infty$ and all other $U_i^\ell = \infty$. Projecting out x_6 yields

$$\texttt{among}((x_3, x_4, x_5), \{1\}, 1, 1)$$

in addition to the original `among` constraints and domains for variables $x_1, \ldots, x_5$. Projecting out x_5 yields

$$\texttt{among}((x_2, x_3, x_4), \{1\}, 1, 1), \quad \texttt{among}((x_3, x_4), \{1\}, 0, 0)$$

in addition to the original constraints on $x_1, \ldots x_4$. The second constraint fixes $x_3 = x_4 = 0$, which makes the first constraint redundant because x_2 has the singleton domain $\{1\}$. Projecting out x_4 yields

$$\texttt{among}((x_1), \{1\}, 1, 1), \quad \texttt{among}((x_1, x_2, x_3), \{1\}, 1, 2),$$
$$\texttt{among}((x_2, x_3), \{1\}, 0, 1), \quad \texttt{among}((x_3), \{1\}, 0, 0)$$

in addition to the original constraints on x_1, x_2, x_3, where the second and third constraints are redundant. This and the domain of x_2 fix $(x_1, x_2, x_3) = (1, 1, 0)$. Projecting out x_3 similarly fixes $(x_1, x_2) = (1, 1)$, and projecting out x_2 fixes $x_1 = 1$.

4.5 Projection of `regular` constraint

The `regular` constraint [43] can be projected onto $x^k = (x_1, \ldots, x_k)$ in a straightforward manner by constructing and truncating the associated state transition graph. The idea is best conveyed by illustration, such as a shift scheduling example from [29]. Suppose we wish to assign a worker to shift a, b, or c on each of 7 days. The worker must work any given shift at least 2 and at most 3 days in a row, and the worker may not transition directly between shifts a and c. In addition to these constraints, the variable domains are initially restricted to $x_1, x_5 \in \{a, c\}$, $x_2 \in \{a, b, c\}$, $x_3, x_6, x_7 \in \{a, b\}$, and $x_4 \in \{b, c\}$. The relevant constraint is $\texttt{regular}(x^7, A)$, where A is the deterministic finite automaton in Fig. 3. A regular expression that encodes the automaton is

$$(((aa|aaa)(bb|bbb))^*|((cc|ccc)(bb|bbb))^*)^*(\epsilon|(aa|aaa)|(cc|ccc))$$

The domains may be filtered by constructing the state transition graph of Fig. 4, where the original domains D_j and the filtered domains D'_j are as shown in the figure. Dashed arcs lead to states that are not backward reachable from accepting states in stage 7 and therefore can be deleted. Each D'_j consists of shifts on solid arcs leading states in stage j of the graph.

A projection onto x^k can be obtained simply by truncating the state transition graph at stage $k+1$ and imposing the resulting graph as a constraint. Such a constraint is easily propagated in a relaxed decision diagram. For example, the projection onto x^3 has 2 solutions, $x^3 = aab$ and ccb, corresponding to the two feasible paths to state 7. If we rely solely on domain filtering, x^3 can take any of 4 values. The worst-case complexity of projecting onto

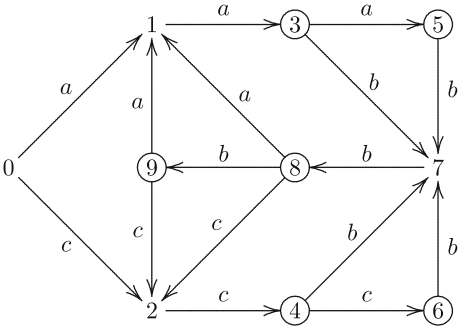

Fig. 3 Finite deterministic automaton for a shift scheduling problem instance. State 0 is the initial state, and accepting states are circled

any x^k is the same as for projecting onto all x^k, namely $\mathcal{O}(nm^2)$, where m is the number of states in the automaton.

4.6 Projection of `alldiff` constraint

Projection of an `all-different` constraint is inherently complicated but tends to simplify when the domains are small. The projection onto $x^k = (x_1, \ldots, x_k)$ takes the form of a disjunction of constraint sets, each of which consists of an `alldiff` constraint and a family of `atmost` constraints. Such a disjunction can be straightforwardly propagated through a relaxed decision diagram. The number of disjuncts can grow quite large in principle, but the disjuncts tend to simplify and/or disappear as variable elimination proceeds, particularly if the domains are small. In practice, one can eliminate variables until the disjunction grows too large, at which point propagating the projection is likely to have little effect in any case.

The projection onto x^k is a disjunction of constraint sets, each of which has the form

$$\texttt{alldiff}(x^k); \quad \texttt{atmost}(x^k, V_i, b_i) \text{ for } i \in I; \quad x_j \in D_j \text{ for } j = 1, \ldots, k \tag{10}$$

where $b_i < k$ for $i \in I$. When $k = n$ there are no `atmost` constraints. The sets V_i are computed as defined below as projection proceeds. Since the projection of a disjunction is the disjunction of the projected disjuncts, it suffices to project each constraint set (10) separately and take the disjunction of the projections.

To simplify matters, we note that $\texttt{atmost}(x^k, V_i, b_i)$ is redundant if the number of variables in x^k whose domains intersect V_i is at most b_i, or in particular if $k \leq b_i$.

The projection of (10) onto x^{k-1} consists of a disjunction of one or more constraint sets having a form similar to (10). We first remove redundant `atmost` constraints from (10).

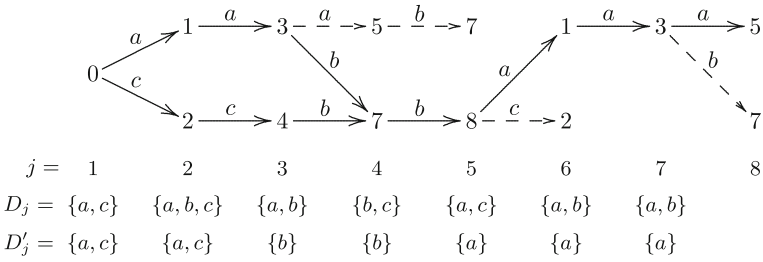

Fig. 4 State transition graph for the shift scheduling problem instance

We then consider several cases, corresponding to possible values of x_k, where each case can give rise to a constraint set.

First we consider cases in which x_k takes a value in V_i for some $i \in I$. In each such case, only $b_i - 1$ values in V_i are available for variables in x^{k-1} to take. So the constraint $\texttt{atmost}(x^k, V_i, b_i)$ becomes $\texttt{atmost}(x^{k-1}, V_i, b_i - 1)$. If $b_i - 1 = 0$, then the values in V_i are forbidden, which means we can delete the atmost constraint and remove the values in V_i from all domains and all sets $V_{i'}$ for $i' \neq i$. The constraints $\texttt{atmost}(x^k, V_{i'}, b_{i'})$ for $i' \neq i$ become $\texttt{atmost}(x^{k-1}, V_{i'}, b_{i'})$, except that we omit redundant constraints.

Now we consider the case in which x_k takes a value that is in none of the sets V_i. Let $R = D_k \setminus \bigcup_{i \in I} V_i$ be the set of x_k's domain values that lie outside these sets. If R is empty, there is no need to consider this case. If R contains at least 2 values, then since x^k is taking a value in R, the variables in x^{k-1} can take at most $|R| - 1$ values in R, and so we impose an additional constraint $\texttt{atmost}(x^{k-1}, R, |R| - 1)$. The atmost constraints in (10) carry over without a change in bounds. If R is a singleton $\{v\}$, then we know x_k takes the value v. This means that v is unavailable in the projection and can be removed from domains $D_1, \ldots, D_{k-1}$ as well as from V_i for all $i \in I$. If a domain becomes empty, then we create no constraint set. Otherwise we carry over the nonredundant atmost constraints in (10). See Algorithm 2 for a precise statement of the procedure. Due to the above argument, we have

Theorem 5 *Algorithm 2 correctly computes the projection of* (10) *onto* x^{k-1}. *Furthermore, the projection of* $\texttt{alldiff}(x)$ *onto* x^k *is a disjunction* $\mathcal{P}$ *of constraint sets of the form* (10). *Thus the projection of* $\texttt{alldiff}(x)$ *onto* x^{k-1} *is the disjunction of all constraint sets generated by applying the algorithm to the disjuncts of* $\mathcal{P}$.

Algorithm 2 Given a projection of $\texttt{alldiff}(x^n)$ onto x^k, this algorithm computes a projection onto x^{k-1}. The projection onto x^k is assumed to be a disjunction of constraint sets, each of which has the form (10). The above algorithm is applied to each disjunct, after which the disjunction of all created constraint sets forms the projection onto x^{k-1}.

For all $i \in I$: if $\texttt{atmost}(x^k, V_i, b_i)$ is redundant then remove i from I.
For all $i \in I$:
  If $D_k \cap V_i \neq \emptyset$ then
    If $b_i > 1$ then
      Create a constraint set consisting of $\texttt{alldiff}(x^{k-1})$,
      $\texttt{atmost}(x^{k-1}, V_{i'}, b_{i'})$ for $i' \in I \setminus \{i\}$, and $\texttt{atmost}\ (x^{k-1}, V_i, b_i - 1)$.
Let $R = D_k \setminus \bigcup_{i \in I} V_i$.
If $|R| > 1$ then
  Create a constraint set consisting of $\texttt{alldiff}(x^{k-1})$,
  $\texttt{atmost}(x^{k-1}, V_{i'}, b_{i'})$ for $i' \in I$, and $\texttt{atmost}\ (x^{k-1}, R, |R| - 1)$.
Else if $|R| = 1$ then
  Let $R = \{v\}$ and remove v from D_j for $j = 1, \ldots, k-1$ and from V_i for $i \in I$.
  If D_j is nonempty for $j = 1, \ldots, k-1$ then
    For all $i' \in I$: if $\texttt{atmost}(x^{k-1}, V_{i'}, b_{i'})$ is redundant then remove i' from i.
    Create a constraint set consisting of $\texttt{alldiff}(x^{k-1})$ and
    $\texttt{atmost}(x^{k-1}, V_{i'}, b_{i'})$ for $i' \in I$.

As an example, suppose we wish to project $\texttt{alldiff}(x^5)$, where the domains $D_1, \ldots, D_5$ are $\{a, b, c\}, \{c, d, e\}, \{d, e, f\}, \{e, f, g\}$, and $\{a, f, g\}$, respectively. To project onto x^4, we note that $R = \{a, f, g\}$, which yields the constraint set

$$\texttt{alldiff}(x^4), \quad \texttt{atmost}(x^4, \{a, f, g\}, 2) \tag{11}$$

When projecting (11) onto x^3, we first consider the case $x_4 \in V_1 = \{a, f, g\}$, which yields the constraint set

$$\texttt{alldiff}(x^3), \quad \texttt{atmost}(x^3, \{a, f, g\}, 1) \tag{12}$$

There is one other case, in which $R = \{e\}$, which allows us to remove e from the domains. We generate the `atmost` constraint $\texttt{atmost}(x^3, \{a, f, g\}, 2)$, which is redundant, yielding the constraint set

$$\texttt{alldiff}(x^3), \quad D_1, \ldots, D_3 = \{a, b, c\}, \{c, d\}, \{d, f\} \tag{13}$$

The projection onto x^3 is the disjunction of (12) and (13).

To project (12) onto x^2, we first consider the case $x_3 \in V_1 = \{a, f, g\}$, which reduces $b_1 = 1$ to zero. This allows us to drop the `atmost` constraint and remove values a, f, g from all domains, leaving the constraint

$$\texttt{alldiff}(x^2), \quad D_1, D_2 = \{b, c\}, \{c, d, e\} \tag{14}$$

In the other case, we have $R = \{d, e\}$, which yields the redundant constraint $\texttt{atmost}(x^2, \{d, e\}, 1)$. The projection onto x^2 is therefore simply $\texttt{alldiff}(x^2)$, and there is no need to compute the projection of (13). Further projection onto x_1 has no effect on the domain of x_1.

5 Concluding remarks

Following George Boole's lead, we have identified projection as a unifying element of optimization, logical inference, and consistency maintenance. We have also begun to explore a research program that addresses inference and constraint satisfaction problems from the perspective of projection, particularly by achieving J-consistency. We showed how to achieve J-consistency for propositional satisfiability, as well as for `among`, `sequence`, `regular`, and `alldiff` constraints.

An obvious next step is to investigate the projection problem for additional global constraints. Even if computing the exact projection is laborious, it may be practical to derive a partial description of the projection, much as solvers often only approximate domain consistency. It is also important to find efficient procedures for propagating projections through a relaxed decision diagram or other structure.

We have not discussed bounds consistency from the perspective of projection, but projection is clearly the essence of the matter. Bounds consistency is most naturally defined when variable domains can be embedded in the real numbers. Then a constraint set S is bounds consistent when the real interval spanned by the domain of each variable x_j is the projection onto x_j of the convex hull of S's satisfaction set. This suggests a natural extension of bounds consistency to an analogue of J-consistency that might be called continuous J-consistency. It is achieved by projecting the convex hull of S's satisfaction set onto x_J.

Projecting a convex hull is closely related to optimization in integer and linear programming [40]. In particular, it is equivalent to finding all valid linear inequalities or cutting

planes over a subset of variables. The identification of cutting planes is a project that has occupied the mathematical programming community for decades. Cutting planes normally serve a different purpose, namely to tighten the continuous relaxation of a set of integer linear inequalities, and the community is generally satisfied to find a few effective cuts, since deriving all valid cuts is generally impractical. Yet a few cuts can partially describe the projection as well as tighten the continuous relaxation.

In fact, cutting planes can serve both purposes in both fields. In CP, they can be propagated through a linear programming relaxation of the problem, which can be effective for bounding the objective function, as well as achieving consistency. In integer programming, they can reduce backtracking even if a continuous relaxation is not used, by improving the degree of consistency of the constraint set, although this phenomenon appeares not to have been investigated.

One difference in the two fields, however, is that cutting plane theory in integer programming focuses on 0–1 variables, as well as general integer variables that appear in inequality constraints, while CP is primarily interested in finite-domain variables that appear in other kinds of combinatorial constraints. In many cases, it is practical to reformulate a problem with 0–1 variables for purposes of relaxation, but ideally one would generate cuts in the original finite-domain variables. Finite-domain cuts have been derived for a few constraints, such as `alldiff` systems (graph coloring) [2, 3, 10, 11, 37, 38], the circuit constraint (which can be used to formulate the traveling salesman problem) [19], and disjunctive scheduling [29]. Yet much remains to be done.

References

1. Andersen, H.R., Hadžić, T., Hooker, J.N., & Tiedemann, P. (2007). A constraint store based on multivalued decision diagrams. In C. Bessiere (Ed.), *Principles and practice of constraint programming (CP 2007). Lecture Notes in Computer Science*, (Vol. 4741 pp. 118–132). Springer.
2. Appa, G., Magos, D., & Mourtos, I. (2004). Linear programming relaxations of multiple all-different predicates. In J.C. Régin, & M. Rueher (Eds.), *CPAIOR 2004 Proceedings. Lecture Notes in Computer Science*, (Vol. 3011 pp. 364–369). Springer.
3. Appa, G., Magos, D., & Mourtos, I. (2004). On the system of two all-different predicates. *Information Processing Letters*, *94*, 99–105.
4. Beame, P., Kautz, H., & Sabharwal, A. (2003). Understanding the power of clause learning. In *International Joint Conference on Artificial Intelligence (IJCAI 2003)*.
5. Beldiceanu, N., & Contejean, E. (1994). Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, *12*, 97–123.
6. Benders, J.F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, *4*, 238–252.
7. Bergman, D., Cire, A., Sabharwal, A., Samulowitz, H., Sarswat, V., & van Hoeve, W.J. (2014). Parallel combinatorial optimization with decision diagrams. In *CPAIOR 2012 Proceedings. LNCS*, (Vol. 8451 pp. 351–367). Springer.
8. Bergman, D., Cire, A.A., van Hoeve, W.J., & Hooker, J.N. Discrete optimization with binary decision diagrams. *INFORMS Journal on Computing* (to appear).
9. Bergman, D., van Hoeve, W.J., & Hooker, J.N. (2011). Manipulating MDD relaxations for combinatorial optimization. In T. Achterberg, & J.S. Beck (Eds.), *CPAIOR 2011 Proceedings. Lecture Notes in Computer Science*, (Vol. 6697 pp. 20–35). Springer.
10. Bergman, D., & Hooker, J.N. (2012). Graph coloring facets from all-different systems. In N. Jussien, & T. Petit (Eds.), *CPAIOR Proceedings* (pp. 50–65). Springer.
11. Bergman, D., & Hooker, J. (2014). Graph coloring inequalities for all-different systems. *Constraints*, *19*, 404–433.
12. Boole, G. (1854). *An investigation of the laws of thought, on which are founded the mathematical theories of logic and Probabilities*. London: Walton and Maberly.

13. Boole, G. (1952). *Studies in logic and probability* (ed. by R. Rhees). La Salle: Open Court Publishing Company.
14. Ciré, A.A., & van Hoeve, W.J. (2012). MDD propagation for disjunctive scheduling. In *Proceedings of the international conference on automated planning and scheduling (ICAPS)* (pp. 11–19). AAAI Press.
15. Cire, A.A., & van Hoeve, W.J. (2013). Multivalued decision diagrams for sequencing problems. *Operations Research*, *61*, 1411–1428.
16. Fourier, L.B.J. (1827). Analyse des travaux de l'Académie Royale des Sciences, pendant l'année 1824, partie mathématique (report of 1824 transactions of the Royal Academy of Sciences, containing Fourier's work on linear inequalities). Histoire de l'Académie Royale des Sciences de l'Institut de France 7, xlvii–iv.
17. Freuder, E.C. (1982). A sufficient condition for backtrack-free search. *Communications of the ACM*, *29*, 24–32.
18. Gebser, M., Kaufmann, B., & Schaub, T. (2009). Solution enumeration for projected boolean search problems. In W.J. van Hoeve, & J.N. Hooker (Eds.), *CPAIOR 2009 Proceedings. Lecture Notes in Computer Science*, (Vol. 5547 pp. 71–86). New York: Springer.
19. Genç-Kaya, L., & Hooker, J.N. (2013). *The Hamiltonian circuit polytope*. Manuscript, Carnegie Mellon University.
20. Hailperin, T. (1976). *Boole's logic and probability, studies in logic and the foundations of mathematics* Vol. 85. North-Holland, Amsterdam.
21. Hansen, P., & Perron, S. (2008). Merging the local and global approaches to probabilistic satisfiability. *International Journal of Approximate Reasoning*, *47*, 125–140.
22. Hoda, S., van Hoeve, W.J., & Hooker, J.N. (2010). A systematic approach to MDD-based constraint programming. In *Proceedings of the 16th international conference on principles and practices of constraint programming. Lecture notes in computer science*. Springer.
23. van Hoeve, W.J., Pesant, G., Rousseau, L.M., & Sabharwal, A. (2006). Revisiting the sequence constraint. In F. Benhamou (Ed.), *Principles and practice of constraint programming (CP 2006). Lecture notes in computer science*, (Vol. 4204 pp. 620–634). Springer.
24. Hooker, J.N. (1988). Generalized resolution and cutting planes. *Annals of Operations Research*, *12*, 217–239.
25. Hooker, J.N. (1988). A mathematical programming model for probabilistic logic. Working paper 05-88-89, Graduate School of Industrial Administration, Carnegie Mellon University.
26. Hooker, J.N. (1992). Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, *6*, 271–286.
27. Hooker, J.N. (1992). Logical inference and polyhedral projection. In *Computer Science Logic Conference (CSL 1991). Lecture Notes in Computer Science*, (Vol. 626 pp. 184–200). Springer.
28. Hooker, J.N. (2007). Planning and scheduling by logic-based Benders decomposition. *Operations Research*, *55*, 588–602.
29. Hooker, J.N. (2012). *Integrated Methods for Optimization*, 2nd edn. Springer.
30. Hooker, J.N., & Ottosson, G. (2003). Logic-based Benders decomposition. *Mathematical Programming*, *96*, 33–60.
31. Hooker, J.N., & Yan, H. (1995). Logic circuit verification by Benders decomposition. In V. Saraswat, & P.V. Hentenryck (Eds.), *Principles and practice of constraint programming: The Newport Papers* (pp. 267–288). Cambridge: MIT Press.
32. Huynh, T., Lassez, C., & Lassez, J.L. (1992). Practical issues on the projection of polyhedral sets. *Annals of Mathematics and Artificial Intelligence*, *6*, 295–315.
33. Jaumard, B., Hansen, P., & Aragão, M.P. (1991). Column generation methods for probabilistic logic. *INFORMS Journal on Computing*, *3*, 135–148.
34. Kavvadias, D., & Papadimitriou, C.H. (1990). A linear prorgamming approach to reasoning about probabilities. *Annals of Mathematics and Artificial Intelligence*, *1*, 189–206.
35. Klinov, P., & Parsia, B. (2011). A hybrid method for probabilistic satisfiability. In N. Bjørner, & V. Sofronie-Stokkermans (Eds.), *23rd International Conference on Automated Deduction (CADE 2011). Lecture Notes in AI*, (Vol. 6803 pp. 354–368). Springer.
36. Klinov, P., & Parsia, B. (2013). Pronto: A practical probabilistic description logic reasoner. In F. Bobillo, P.C.G. Costa, C. d'Amato, N. Fanizzi, K.B. Laskey, K.J. Laskey, T. Lukasiewicz, M. Nickles, & M. Pool (Eds.), *Uncertainty reasoning for the Semantic Web II (URSW 2008–2010) LNAI*, (Vol. 7123 pp. 59–79). Springer.
37. Magos, D., & Mourtos, I. (2011). On the facial structure of the alldifferent system. *SIAM Journal on Discrete Mathematics*, 130–158.
38. Magos, D., Mourtos, I., & Appa, G. (2012). A polyhedral approach to the *alldifferent* system. *Mathematical Programming*, *132*, 209–260.

39. Maher, M.J., Narodytska, N., Quimper, C.G., & Walsh, T. (2008). Flow-based propagators for the SEQUENCE and related global constraints. In P.J. Stuckey (Ed.), *Principles and Practice of Constraint Programming (CP 2008). Lecture Notes in Computer Science*, (Vol. 5202 pp. 159–174). Springer.
40. Martin, R.K. (1999). *Large scale linear and integer optimization: A unified approach*. New York: Springer.
41. Motzkin, T.S. (1983). Beiträge zur Theorie der linearen Ungleichungen. Ph.D. thesis, University of Basel (1936), English translation: Contributions to the theory of linear inequalities, RAND Corporation Translation 22, Santa Monica, CA (1952), reprinted in D. Cantor, B. Gordon and B. Rothschild, eds., Theodore S. Motzkin: Selected Papers, Birkhäuser, Boston, 1–80.
42. Nilsson, N.J. (1986). Probabilistic logic. *Artificial Intelligence*, *28*, 71–87.
43. Pesant, G. (2004). A regular language membership constraint for finite sequences of variables. In M. Wallace (Ed.), *Principles and practice of constraint programming (CP 2004). Lecture Notes in Computer Science*, (Vol. 3258 pp. 482–495). Springer.
44. Quine, W.V. (1952). The problem of simplifying truth functions. *American Mathematical Monthly*, *59*, 521–531.
45. Quine, W.V. (1955). A way to simplify truth functions. *American Mathematical Monthly*, *62*, 627–631.
46. Régin, J.C., & Puget, J.F. (1997). A filtering algorithm for global sequencing constraints. In G. Smolka (Ed.), *Principles and practice of constraint programming (CP 1997). Lecture Notes in Computer Science*, (Vol. 3011 pp. 32–46). Springer.
47. Williams, H.P. (1987). Linear and integer programming applied to the propositional calculus. *International Journal of Systems Research and Information Science*, *2*, 81–100.
48. Williams, H.P., & Hooker, J.N. (2014). Integer programming as projection. Working paper LSEOR 13.143, London School of Economics.