Projection, Inference, and Consistency

J. N. Hooker
Carnegie Mellon University, Pittsburgh, USA
jh38@andrew.cmu.edu

Abstract

Projection can be seem as a unifying concept that underlies inference in logic and consistency maintenance in constraint programming. This perspective allows one to import projection methods into both areas, resulting in deeper insight as well as faster solution methods. We show that inference in propositional logic can be achieved by Benders decomposition, an optimization method based on projection. In constraint programming, viewing consistency maintenance as projection suggests a new but natural concept of consistency that is achieved by projection onto a subset of variables. We show how to solve this combinatorial projection problem for some global constraints frequently used in constraint programming. The resulting projections are useful when propagated through decision diagrams rather than the traditional domain store.

1 Introduction

Projection can be seem as a unifying concept that underlies both inference in logic and consistency maintenance in constraint programming. Projection methods that have been developed in other contexts can therefore be harnessed to help solve inference and constraint programming problems.

In propositional logic, for example, inference can be conceived as the general problem of deducing all information that can be expressed in terms of a specified subset of variables (atomic propositions). This can also be understood as a projection problem. It can be solved not only by the resolution method, which is closely parallel to a classical projection method for linear inequalities (Fourier-Motzkin elimination), but by *Benders decomposition*, an optimization method that may seem unrelated to inference. It is well known that the Benders method can compute a projection onto a subset of variables. An extension of the classical method, *logic-based* Benders decomposition, solves the inference problem for propositional logic. Moreover, it can take advantage of clause learning techniques used in state-of-the-art propositional satisfiability (SAT) solvers.

Consistency maintenance is a fundamental tool of constraint programming. Its purpose is to exclude assignments of values to variables that are inconsistent with any feasible solution of a constraint, thereby reducing the amount of search necessary to find a solution. The results of consistency maintenance for one constraint are *propagated* to other constraints through some kind of data structure, such as a *domain store*, which consists of the set of possible values for each individual variable.

Consistency maintenance is both an inference problem and a projection problem. It is an inference problem because it deduces constraints that variable assignments must satisfy. It is a projection problem because excluding infeasible assignments to a subset of variables is equivalent to computing the projection of the feasible set onto those variables.

This unifying perspective can be exploited in constraint programming by addressing consistency maintenance explicitly as a projection problem. Existing types of consistency are already forms of projection, including domain consistency, bounds consistency, and k-consistency. However, viewing consistency in this light suggests a simple type of consistency that has not been investigated. We call it J*-consistency*, which is achieved by projecting the problem's solution set onto a subset J of variables. Achieving J-consistency can reduce backtracking when the solver propagates through a richer data structure than a domain store.

This research program poses a problem that might be called *combinatorial projection*: projecting a combinatorial constraint, such as the global constraints routinely used in constraint programming, onto a subset of variables. We solve this problem here for a small collection of standard global constraints: `among`, `sequence`, `regular`, and `all-different`.

2 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.

Table 1: A set of logical clauses.

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	$\vee\, \bar{x}_4$	$\vee\, \bar{x}_5$	

2.1 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.

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). 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, let S consist 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. (In fact, they are the prime implicates of S.)

2.2 Inference for Linear Inequalities

The classical projection method for a system $Ax \geq b$ of linear inequalities is *Fourier-Motzkin elimination*. We can compute the projection onto $x_1, \ldots, x_k$ by eliminating variables $x_n, x_{n-1}, \ldots, x_{k+1}$ one at a time. Let S initially be the set of inequalities $Ax \geq b$. Each variable x_j is eliminated as follows. For each pair of inequalities in S that have the form $c\bar{x} + c_0 x_j \geq \gamma$ and $d\bar{x} - d_0 x_j \geq \delta$, where $c_0, d_0 > 0$ and $\bar{x} = (x_1, \ldots, x_{j-1})$, we have

$$-\frac{c}{c_0}\bar{x} + \frac{\gamma}{c_0} \leq x_j \leq \frac{d}{d_0}\bar{x} - \frac{\delta}{d_0}$$

or $L \leq x_j \leq U$ 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 x_j. The inequalities in S at the end of the procedure describe the projection and therefore capture everything that can be inferred from $Ax \geq b$ in terms of $x_1, \ldots, x_k$.

Fourier-Motzkin elimination can also be used to compute inferences in probability logic, which can formulated as an optimization problem with linear inequality constraints [Boole, 1854; Hailperin, 1976; Nilsson, 1986]. It is more efficient, however, to solve the problem with modern linear programming (LP) methods that use column generation [Hansen and Perron, 2008; Hooker, 1988b; Jaumard *et al.*, 1991; Klinov and Parsia, 2013].

2.3 Inference in Propositional Logic

An elimination procedure based on resolution [Quine, 1952; 1955] 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 $C \vee D$. Resolution can in fact be seen as a form of Fourier-Motzkin elimination plus rounding [Williams, 1987]. It can be shown [Hooker, 1992b; 2012] that eliminating variables x_j for $j \notin J$ by resolution (in any order) yields a projection of S onto x_J.

Resolution tends to be impractical, but Benders decomposition [Benders, 1962] can be much more efficient, especially when J is small. The classical Benders method applies only to problems with an LP subproblem, but we use logic-based Benders decomposition, which is suitable for general constraint solving and optimization [Hooker, 2000; 2007; 2012; Hooker and Ottosson, 2003].

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 *Benders cut* or nogood clause 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 $\bar{x}_J$. 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 [Hooker, 2012].

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 [Beame *et al.*, 2003] 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. 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 $\{x_1 \vee x_2, x_1 \vee x_3\}$.

Other adaptations of resolution and Fourier-Motzkin elimination can be used to compute projections for cardinality clauses [Hooker, 1988a], 0–1 linear inequalities [Hooker, 1992a], and general integer linear inequalities [Williams and Hooker, 2014].

3 Consistency Maintenance

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 $j = 1, \dots, n$. Maintaining domain consistency (or an approximation of it) for some individual constraints in S, and propagating the reduced domains through a domain store, 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 by projecting only subsets of constraints over k variables onto subsets of $k-1$ variables [Freuder, 1982].

3.1 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 $j = 1, \dots, n$.

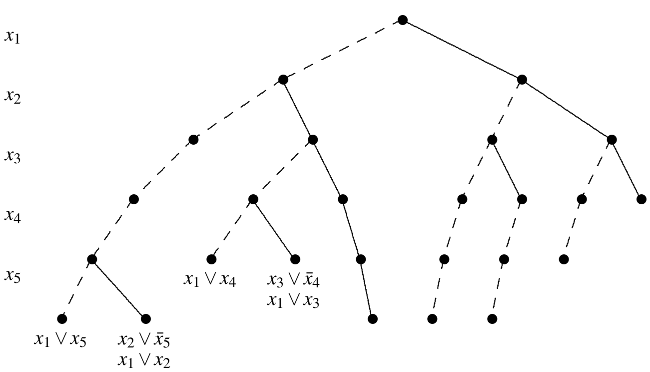


Figure 1: Branching tree for a SAT instance. Dashed arcs indicate $x_j = \mathrm{F}$ and solid arcs $x_j = \mathrm{T}$. Conflict clauses are shown at failure nodes. Solutions are found at remaining leaf nodes, from which the search backtracks to x_3.

If we branch on variables in the order $x_1, \dots, x_n$, a natural strategy is to project out variables in reverse order $x_n, x_{n-1}, \dots$ 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.

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 [Bergman *et al.*, 2014; 2016; 2011; Ciré and van Hoeve, 2012; Cire and van Hoeve, 2013]. Maintaining J-consistency could have a significant effect on propagation in this context. This is illustrated in [Hooker, 2016].

3.2 Projection of Among Constraint

Projecting out variables in an `among` constraint [Beldiceanu and Contejean, 1994] is relatively 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 $\alpha^+ = \max\{\alpha, 0\}$, and assume $0 \le \ell \le u \le n$,

Theorem 2 *The projection of* $\texttt{among}(x, V, \ell, u)$ *onto* $\bar{x} = (x_1, \dots, x_{n-1})$ *is among*$(\bar{x}, V, \ell', u')$*, where*

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

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.

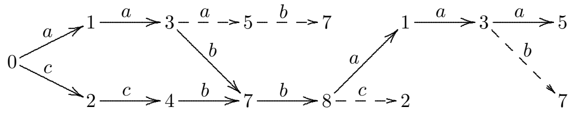
3.3 Projection of Sequence Constraint

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.

We assume without loss of generality that the `sequence` constraint applies to 0-1 variables $x_1, \dots, x_n$ [van Hoeve *et al.*, 2006; Régin and Puget, 1997]. It enforces overlapping constraints of the form

$$\texttt{among}((x_{\ell-q+1}, \dots, x_\ell), \{1\}, L_\ell, U_\ell) \quad (1)$$

for $\ell = q, \dots, n$, 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 $x_n, \dots, x_1$:



$j=$	1	2	3	4	5	6	7	8
$D_j=$	$\{a,c\}$	$\{a,b,c\}$	$\{a,b\}$	$\{b,c\}$	$\{a,c\}$	$\{a,b\}$	$\{a,b\}$	
$D'_j=$	$\{a,c\}$	$\{a,c\}$	$\{b\}$	$\{b\}$	$\{a\}$	$\{a\}$	$\{a\}$	

Figure 2: State transition graph for a shift scheduling problem instance. D_j is the original domain of x_j, and D'_j result of achieving domain consistency. States 3, 4, 5, and 8 are valid terminal states in the automaton. Dashed lines lead to nonterminal states that are infeasible because there are no out-transitions consistent with the given variable domains.

Theorem 3 *Given any $k \in \{0,\ldots,n\}$, the projection of the* `sequence` *constraint defined by (1) onto $(x_1,\ldots,x_k)$ is described by a generalized* `sequence` *constraint that enforces constraints of the form*

$$\texttt{among}\Big((x_i,\ldots,x_\ell),\{1\},L^\ell_{\ell-i+1},U^\ell_{\ell-i+1}\Big) \tag{2}$$

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 (2) 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*

$$\hat{L}^\ell_i = \begin{cases}\max\{L^\ell_i, L^k_{i+k-\ell}-U^k_{k-\ell}\}, i=1,\ldots,q-k+\ell,\\ L^\ell_i,\ \textit{for}\ i=q-k+\ell+1,\ldots,q\end{cases}$$

$$\hat{U}^\ell_i = \begin{cases}\min\{U^\ell_i, U^k_{i+k-\ell}-L^k_{k-\ell}\}, i=1,\ldots,q-k+\ell,\\ U^\ell_i,\ \textit{for}\ i=q-k+\ell+1,\ldots,q\end{cases}$$

The worst-case complexity of projecting out each variable x_k is $\mathcal{O}(kq)$.

3.4 Projection of Regular Constraint

The `regular` constraint [Pesant, 2004] formulates scheduling and related constraints as regular expressions. Projection can be carried out by constructing and truncating the state transition graph for the associated deterministic finite automaton. For example, the regular expression

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

represents a shift scheduling problem and generates the transition graph of Fig. 2 over a 7-day period, where a,b,c are shifts and x_j is the assigned shift for day j. The graph shows 2 feasible shift assignments: *aabbaaa* and *ccbbaaa*. Truncating the graph at stage $j=4$ yields a projection onto (x_1,x_2,x_3), which has the two feasible solutions *aab* and *ccb*. Details may be found in [Hooker, 2016].

3.5 Projection of All-different Constraint

The constraint `alldiff`(x) requires that $x_1,\ldots,x_n$ take different values. While domain consistency is relatively easy to achieve for the constraint, using a matching algorithm, general projection is surprisingly complex. 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. 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.

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

$$\begin{array}{l}\texttt{alldiff}(x^k);\ \ \texttt{atmost}(x^k,V_i,b_i)\ \text{for}\ i\in I;\\ x_j\in D_j\ \text{for}\ j=1,\ldots,k\end{array} \tag{3}$$

where $b_i < k$ for $i \in I$. The `atmost` constraint says that at most b_i occurrences of values in V_i appear in x^k. When $k=n$ there are no `atmost` constraints. We also note that `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 $k \le b_i$. Algorithm 1 is applied to compute the projection onto $x^{n-1},\ldots,x^k$ sequentially.

Theorem 4 *Algorithm 1 correctly computes the projection of (3) onto x^{k-1}.*

Algorithm 1 Given a projection of alldiff(x^n) onto x^k, compute 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 (3). The 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 `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 `alldiff`(x^{k-1}),
      `atmost`$(x^{k-1},V_{i'},b_{i'})$ for $i' \in I\setminus\{i\}$, and
      `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 `alldiff`(x^{k-1}),
  `atmost`$(x^{k-1},V_{i'},b_{i'})$ for $i' \in I$, and
    `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 `atmost`$(x^{k-1},V_{i'},b_{i'})$ is redundant
      then remove i' from i.
    Create a constraint set consisting of `alldiff`(x^{k-1}) and
    `atmost`$(x^{k-1},V_{i'},b_{i'})$ for $i' \in I$.

As an example, suppose we wish to project `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. The projection onto x^4 is

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

The projection onto x^3 is the disjunction of the following two constraint sets:

$$\texttt{alldiff}(x^3),\ \ \texttt{atmost}(x^3,\{a,f,g\},1)$$
$$\texttt{alldiff}(x^3),\ \ D_1,\ldots,D_3=\{a,b,c\},\{c,d\},\{d,f\}$$

References

[Beame *et al.*, 2003] P. Beame, H. Kautz, and A. Sabharwal. Understanding the power of clause learning. In *International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 2003.

[Beldiceanu and Contejean, 1994] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathematical and Computer Modelling*, 12:97–123, 1994.

[Benders, 1962] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[Bergman *et al.*, 2011] D. Bergman, W.-J. van Hoeve, and J. N. Hooker. Manipulating MDD relaxations for combinatorial optimization. In *CPAIOR 2011 Proceedings*, pages 20–35, 2011.

[Bergman *et al.*, 2014] D. Bergman, A. Cire, A. Sabharwal, H. Samulowitz, V. Sarswat, and W.-J. van Hoeve. Parallel combinatorial optimization with decision diagrams. In *CPAIOR 2012 Proceedings*, pages 351–367, 2014.

[Bergman *et al.*, 2016] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Discrete optimization with binary decision diagrams. *INFORMS Journal on Computing*, 28:47–66, 2016.

[Boole, 1854] G. Boole. *An Investigation of the Laws of Thought, On Which are Founded the Mathematical Theories of Logic and Probabilities*. Walton and Maberly, London, 1854.

[Ciré and van Hoeve, 2012] A. A. Ciré and W.-J. van Hoeve. MDD propagation for disjunctive scheduling. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pages 11–19, 2012.

[Cire and van Hoeve, 2013] A. A. Cire and W.-J. van Hoeve. Multivalued decision diagrams for sequencing problems. *Operations Research*, 61:1411–1428, 2013.

[Freuder, 1982] E. C. Freuder. A sufficient condition for backtrack-free search. *Communications of the ACM*, 29:24–32, 1982.

[Hailperin, 1976] T. Hailperin. *Boole's Logic and Probability*, volume 85 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1976.

[Hansen and Perron, 2008] P. Hansen and S. Perron. Merging the local and global approaches to probabilistic satisfiability. *International Journal of Approximate Reasoning*, 47:125–140, 2008.

[Hooker and Ottosson, 2003] J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

[Hooker, 1988a] J. N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12:217–239, 1988.

[Hooker, 1988b] J. N. Hooker. A mathematical programming model for probabilistic logic. Working paper 05-88-89, Graduate School of Industrial Administration, Carnegie Mellon University, 1988.

[Hooker, 1992a] J. N. Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6:271–286, 1992.

[Hooker, 1992b] J. N. Hooker. Logical inference and polyhedral projection. In *Computer Science Logic Conference (CSL 1991)*, volume 626 of *Lecture Notes in Computer Science*, pages 184–200. Springer, 1992.

[Hooker, 2000] J. N. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. Wiley, New York, 2000.

[Hooker, 2007] J. N. Hooker. Planning and scheduling by logic-based Benders decomposition. *Operations Research*, 55:588–602, 2007.

[Hooker, 2012] J. N. Hooker. *Integrated Methods for Optimization, 2nd ed.* Springer, 2012.

[Hooker, 2016] J. N. Hooker. Projection, consistency, and George Boole. *Constraints*, 21:59–76, 2016.

[Jaumard *et al.*, 1991] B. Jaumard, P. Hansen, and M. P. Aragão. Column generation methods for probabilistic logic. *INFORMS Journal on Computing*, 3:135–148, 1991.

[Klinov and Parsia, 2013] P. Klinov and B. Parsia. Pronto: A practical probabilistic description logic reasoner. In *Uncertainty Reasoning for the Semantic Web II (URSW 2008–2010)*, pages 59–79, 2013.

[Nilsson, 1986] N. J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–87, 1986.

[Pesant, 2004] G. Pesant. A regular language membership constraint for finite sequences of variables. In *Principles and Practice of Constraint Programming (CP 2004)*, pages 482–495, 2004.

[Quine, 1952] W. V. Quine. The problem of simplifying truth functions. *American Mathematical Monthly*, 59:521–531, 1952.

[Quine, 1955] W. V. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62:627–631, 1955.

[Régin and Puget, 1997] J.-C. Régin and J.-F. Puget. A filtering algorithm for global sequencing constraints. In *Principles and Practice of Constraint Programming (CP 1997)*, pages 32–46, 1997.

[van Hoeve *et al.*, 2006] W.-J. van Hoeve, G. Pesant, L.-M. Rousseau, and A. Sabharwal. Revisiting the sequence constraint. In *Principles and Practice of Constraint Programming (CP 2006)*, pages 620–634, 2006.

[Williams and Hooker, 2014] H. P. Williams and J. N. Hooker. Integer programming as projection. Working paper LSEOR 13.143, London School of Economics, 2014.

[Williams, 1987] H. P. Williams. Linear and integer programming applied to the propositional calculus. *International Journal of Systems Research and Information Science*, 2:81–100, 1987.