Projection, Inference, and Consistency

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

Abstract

Projection can be seen 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 seen 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 x1, ..., xn, and that x1, ..., xk are relevant to product P. For example, x1 may be the model of P produced, x2 the output level of P, x2 its unit manufacturing cost, and so forth up to xk. Then we wish to deduce from S all constraints containing x1, ..., xk. We will see that this is a projection problem.
Table 1: A set of logical clauses.

| x_1  | \lor x_4 \lor x_5 |
| x_1  | \lor x_2 \lor x_5 |
| x_1  | \lor x_2 \lor x_6 |
| x_1  | \lor x_5 \lor x_6 |
| x_2  | \lor x_5 \lor x_6 |
| x_2  | \lor \neg x_5 \lor x_6 |
| x_3  | \lor x_4 \lor x_5 |
| x_3  | \lor x_4 \lor x_6 |

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 \( V' \subset D_J \) of tuples is the projection onto \( x_J \) of \( V \subset D \) when \( V' = \{v_J \mid v \in V\} \). This can be written \( V' = V\mid J \). 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 constraint only when it fixes all variables in the constraint.

Let \( D_J(S) \) be the set of all \( v \in D_J \) for which \( x_J = v \) satisfies \( S \) (i.e., satisfies all the constraints in \( S \)). We say that \( S \) is a constraint set over \( x \) when it contains only variables in \( x = (x_1, \ldots, x_n) \), 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, \( D_J(S') = D_j(S) \mid J \). It is easy to show that projection captures exactly what \( S \) implies 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 \( S' = \{x_1 \lor x_2, x_1 \lor x_3\} \) 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].
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 \( \{ x_1 \lor x_2, x_1 \lor x_3 \} \).

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 \( j = 1, \ldots, n \). Maintaining 
domain consistency (or an approximation of it) for some indi-
vidual 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 project-
ing 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 \( j = 1, \ldots, n \).

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 ex-
cessive. 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 individ-
ual constraints when projection 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 vari-
able elimination yields another among constraint. If \( x = 
(x_1, \ldots, x_n) \), the constraint among \( (x, V, \ell, u) \) requires that 
at least \( \ell \) 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 \leq \ell \leq u \leq n \,

**Theorem 2** The projection of among \( (x, V, \ell, u) \) onto \( \bar{x} 
= (x_1, \ldots, x_{n-1}) \) is among \( (\bar{x}, V, \ell', u') \), where 
\[
(\ell', u') = \begin{cases} 
(\ell + 1, u - 1), & \text{if } D_n \subseteq V 

(\ell, \min\{u, n - 1\}), & \text{if } D_n \cap V = \emptyset 

((\ell - 1)^+, \min\{u, n - 1\}), & \text{otherwise}
\end{cases}
\]

Variables \( x_n, x_{n-1}, \ldots, x_1 \) are projected out sequentially 
by applying the theorem recursively. The original constraint 
is feasible if and only if \( \ell' \leq u' \) 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 con-
straint has an integrality property that makes a polyhedral 
projection technique adequate, and Fourier-Motzkin simpli-
fies 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 \( x_1, \ldots, x_n \) [van Hoeve et 
al., 2006; Régin and Puget, 1997]. It enforces overlapping 
constraints of the form 
\[
\text{among}(\{x_{\ell-q+1}, \ldots, x_{\ell}\}, \{1\}, L_{\ell}, U_{\ell})
\]
for \( \ell = q, \ldots, n \), where \( L_{\ell}, U_{\ell} \) are nonnegative integers, 
and where domain \( D_j \) is defined by \( \alpha_j \leq x_j \leq \beta_j \) for 
\( \alpha_j, \beta_j \in \{ 0, 1 \} \). Note that we allow different bounds for 
different positions \( \ell \) in the sequence. The following theorem 
provides a recursion for eliminating \( x_n, \ldots, x_1 \):
The general projection is surprisingly complex. The projection different values. While domain consistency is relatively easy

domination can be carried out by constructing and truncating the

constraint onto.

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 (I) onto \( (x_1, \ldots, x_k) \)
is described by a generalized sequence constraint that enforces
constraints of the form

\[
\bigcup \left( (x_1, \ldots, x_I), \{1\}, L^{\ell}_{i+1}, U^{\ell}_{i+1} \right)
\]

(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 \( L^{\ell}_{i+1} \)
replaced by \( \hat{L}^{\ell}_{i+1} \) and \( U^{\ell}_{i+1} \) by \( \hat{U}^{\ell}_{i+1} \), where

\[
\hat{L}^{\ell}_{i} = \begin{cases} 
\max\{L^{\ell}_{i}, l^{k}_{i+k-\ell} - U^{k}_{k-\ell} \}, & i = 1, \ldots, q - k + \ell, \\
L^{\ell}_{i}, & i = q - k + \ell + 1, \ldots, q 
\end{cases}
\]

\[
\hat{U}^{\ell}_{i} = \begin{cases} 
\min\{U^{\ell}_{i}, U^{k}_{i+k-\ell} - l^{k}_{k-\ell} \}, & i = 1, \ldots, q - k + \ell, \\
U^{\ell}_{i}, & i = q - k + \ell + 1, \ldots, q 
\end{cases}
\]

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

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

\[
((\text{aa})\text{aaa})(\text{bb})\text{bbb})^{+}(\text{cc})\text{ccc}(\text{bb})\text{bbb})^{+}(\text{cc})\text{ccc})
\]

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: \( aabbaa \) and \( ccbbaa \).
Truncating the graph at stage \( j = 4 \) yields a projection onto
\( (x_1, x_2, x_3) \), which has two feasible solutions \( aab \) and \( ccb \).
Details may be found in [Hooker, 2016].

3.5 Projection of All-different Constraint

The constraint \( \text{alldiff}(x) \) 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 \( \text{alldiff} \)
constraint and a family of \( \text{atmost} \) 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,

\[
\text{alldiff}(x^k); \text{atmost}(x^k, V_i, b_i) \text{ for } i \in I;
\]

(3)

where \( b_i < k \) for \( i \in I \). The \text{atmost} constraint says that
at most \( b_i \) occurrences of values in \( V_i \) appear in \( x^k \). When
\( k = n \) there are no \text{atmost} constraints. We also note that
\text{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 \( k \leq b_i \). 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 \( \text{alldiff}(x^n) \) 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 \( \text{atmost}(x^k, V_i, b_i) \) is redundant then
remove \( i \) from \( I \).

For all \( i \in I \):
If \( D_i \cap V_i \neq \emptyset \) then
If \( b_i > 1 \) then
Create a constraint set consisting of \( \text{alldiff}(x^{k-1}), \text{atmost}(x^{k-1}, V_i, b_i) \text{ for } i' \in I \setminus \{i\} \), and
\text{atmost}(x^{k-1}, V_i, b_i - 1).

Let \( R = D_i \setminus \bigcup_{i' \in I} \{i'\} \).
If |\( R \}| > 1 then
Create a constraint set consisting of \( \text{alldiff}(x^{k-1}), \text{atmost}(x^{k-1}, V_i, b_i) \text{ for } i' \in I \), and
\text{atmost}(x^{k-1}, R, |\( R \}| - 1).

Else if |\( R \}| = 1 then
Let \( R = \{i\} \) 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 \( \text{atmost}(x^{k-1}, V_i, b_i) \) is redundant
then remove \( i' \) from \( i \).
Create a constraint set consisting of \( \text{alldiff}(x^{k-1}) \) and
\text{atmost}(x^{k-1}, V_i, b_i) \text{ for } i' \in I \).

As an example, suppose we wish to project \( \text{alldiff}(x^5) \),
where the domains \( D_1, \ldots, D_5 \) are \( \{a, b, c\}, \{d, e, f\}, \{c, d, e\}, \{d, e, f\}, \{e, f, g\}, \) and \( \{a, f, g\} \), respectively.
The projection onto \( x^4 \) is

\[
\text{alldiff}(x^4); \text{atmost}(x^4, \{a, f, g\}, 2)
\]

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

\[
\text{alldiff}(x^3), \text{atmost}(x^3, \{a, f, g\}, 1)
\]

\[
\text{alldiff}(x^3), D_1, \ldots, D_3 = \{a, b, c\}, \{c, d\}, \{d, f\}
\]
References


