Efficient Representation of Adhoc Constraints *

Kenil C.K. Cheng and Jimmy H.M. Lee
Dept of Comp. Sci. & Eng.
The Chinese University of Hong Kong, Hong Kong
{ckcheng, jlee}@cse.cuhk.edu.hk

Peter J. Stuckey
Dept. of Comp. Sci. & Soft. Eng.
University of Melbourne, Australia
pjs@cs.mu.oz.au

1 Introduction

Constraint programming is a promising technique for solving many difficult combinatorial problems. Since real-life con-
straints can be difficult to describe in symbolic expressions, or provide very weak propagation from their symbolic repre-
sentation, they are sometimes represented in the form of the sets of solutions or sets of nogoods. This adhoc represen-
tation provides strong propagation through domain (general-
ized arc) consistency techniques. However, the adhoc repre-
sentation is expensive in terms of memory and computation, when the adhoc constraint is large.

So there is interest in determining less expensive meth-
ods for building propagators for adhoc constraints [Frühwirth, 1998; Apt and Monfroy, 1999; Abdennadher and Rigotti,
2000; Barták, 2001; Dao et al., 2002].

In this paper, we propose a new language-independent re-
presentation for adhoc constraints, the box constraint collec-
tion. The idea is to break up an adhoc constraint into pieces and cover these pieces using box constraints as tiles. This can be done automatically with a greedy algorithm. With the aid of constructive disjunction and a suitable choice of constraint templates in the collection, our new representation achieves domain consistency.

2 Propagation Based Constraint Solving

In this section we give our terminology for constraint satis-
faction problems, and propagation based constraint solving.

An integer valuation \( \theta \) is a mapping of variables to integer values, written \( \{ x_1 \mapsto d_1, \ldots, x_n \mapsto d_n \} \). We extend the valuation \( \theta \) to map expressions and constraints involving the variables in the natural way. We sometimes treat a valuation \( \theta = \{ x_1 \mapsto d_1, \ldots, x_n \mapsto d_n \} \) as the constraint \( x_1 = d_1 \land \ldots \land x_n = d_n \). Let \( \text{vars} \) be the function that returns the set of (free) variables appearing in a constraint or valuation.

A domain \( D \) is a complete mapping from a fixed (countable) set of variables \( V \) to finite sets of integers. A domain \( D_1 \) is stronger than a domain \( D_2 \), written \( D_1 \subseteq D_2 \), if \( D_1(x) \subseteq D_2(x) \) for all variables \( x \).

In an abuse of notation, we define a valuation \( \theta \) to be an element of a (non-false) domain \( D \), written \( \theta \in D \), if \( \theta(x) \in D(x) \) for all \( x \in \text{vars}(\theta) \).

We are also interested in the notion of an initial domain, denoted by \( D_{\text{inst}} \). The initial domain gives the initial values possible for each variable.

A constraint \( c \) over variables \( x_1, \ldots, x_n \), written as \( c(x_1, \ldots, x_n) \), restricts the values that each variable \( x_i \) can take simultaneously. An adhoc constraint \( c(x_1, \ldots, x_n) \) is defined extensionally as a set of valuations \( \theta \) over the vari-
ables \( x_1, \ldots, x_n \). We say \( \theta \in c \) is a solution of \( c \). For any valuation \( \theta \) on variables \( x_1, \ldots, x_n \), with \( \theta \not\in c \), we call \( \theta \) a nogood of \( c \).

Often we define constraints intensionally using some well
understood mathematical syntax. For an intensionally defined constraint \( c \) we have that \( \theta \in c \) if \( \text{vars}(\theta) = \text{vars}(c) \land \models \theta \models c \). For example the constraint \( x_1 = x_2 + 1 \) where \( D_{\text{inst}}(x_1) = \{ 1, 2, 3 \} \) defines the set of solutions \( \{ x_1 \mapsto 2, x_2 \mapsto 1 \}, \{ x_1 \mapsto 3, x_2 \mapsto 2 \} \).

A constraint satisfaction problem (CSP) [Tsang, 1993], consists of a set of constraints \( c_1, \ldots, c_k \) over a set of vari-
ables \( x_1, \ldots, x_n \), where each variable \( x_i \) can only take values from its domain \( D_{\text{inst}}(x_i) \), a set of integers. Solving a CSP requires finding a value for each variable from its domain so that no constraint is violated, i.e. all constraints are satisfied.

A propagator \( f \) is a monotonically decreasing function from domains to domains. The generalized arc consistent propagator for a constraint \( c \) is defined as \( \text{dom}(c)(D)(x) = \{ \theta(x) \mid \theta \in D \land \theta \models c \} \) where \( x \in \text{vars}(c) \), otherwise \( \text{dom}(c)(D)(x) = D(x) \). A propagation solver for propa-
gators \( F \) repeatedly applies propagators \( f \in F \) to a domain \( D \) until no further change in \( D \) results.

3 Box Constraint Collections

Adhoc constraints are usually implemented as tabled con-
straints by listing all the solutions or nogoods, incurring space and time overhead. Often we represent a constraint in an ad-
hoc manner because it is difficult (or unwieldy) to describe it using a symbolic expression. However, it may be easier to find symbolic expressions if we examine part of the so-
lution space. Therefore, we propose representing an adhoc constraint \( c_{\text{adhoc}} \) with a set of simple constraints in DNF.

A box \( B = \prod_{i=1}^n [a_i..b_i] \) is an n-dimensional hyper-cube, where \( [a_i..b_i] \) is a (closed) interval of integers \( a_i \) and \( b_i \). If
\( c(x_1, \ldots, x_n) \) is a constraint on variables \( x_1, \ldots, x_n \), then 
\[ \bigwedge_{j=1}^n a_{ij} \leq x_j \leq b_{ij} \land c(x_1, \ldots, x_n) \] is a box constraint, which we write as \( B \Rightarrow c \). We restrict the form of constraints \( c(x_1, \ldots, x_n) \) to certain templates. A box constraint collection (BCC) is simply a disjunction of box constraints.

The idea is thus to use box constraints in a collection as “tiles” to cover the solution space of an adhoc constraint. The template defining \( c \) in a box constraint \( B \Rightarrow c \) determines the shape of the tile. Triangles and rectangular boxes are good tile shapes for filling grids. If \( c \) is true, then \( B \Rightarrow c \) is simply the box \( B \). If \( c \) is of the form \( \sum_{j=1}^n a_{ij} x_j \leq a_0 \), then we call \( B \Rightarrow c \) a triangle.

Lemma 1

Let

\[ c_{\text{adhoc}}(x_1, \ldots, x_n) \equiv \bigvee_{i=1}^m B_i \Rightarrow c_i(x_1, \ldots, x_n) \]

and suppose each constraint \( c_i \) is implemented by a generalized arc consistent propagator, then using constructive disjunction on this representation achieves generalized arc consistency for \( c_{\text{adhoc}} \).

\[ \text{Example 1} \]

A box constraint collection representation of the constraint \( c_{\text{tri}} \) shown in Figure 1 is

\[ [1..3] \times [1..3] \Rightarrow X + Y \geq 4 \lor [3..5] \times [3..5] \Rightarrow X + Y \leq 8 \]

Due to space limitation, we cannot show how box constraint collections can be compiled into indexicals directly and efficiently.

\[ \text{4 Experiments} \]

We compare the propagation efficiency among box (indexical BCCs for boxes only), tri-box (indexical BCCs for triangles and boxes) and rel (the built-in relation/3 for binary adhoc constraints) on randomly generated cubic inequalities of the form \( d_1 X^3 + d_2 X^2 Y + d_3 X Y^2 + d_4 Y^3 + d_5 X^2 + d_6 X Y + d_7 Y^2 + d_8 X + d_9 Y \leq d_{10} \). The coefficients are randomly chosen between \([-9..9]\). The domain size is 100. For each variable \( X \) and \( Y \), we repeat \( M \) times picking a subset \( S \subseteq D_{\text{rel}}(x) \) where \( |S| = W \), and adding the constraints \( x \neq v \) for each \( v \in S \). These constraint additions are then removed and the next set \( S \) picked. We do our implementation with SICStus Prolog 3.9.1 on a Sun Blade 1000 workstation.

Table 1 summarizes some results. \( N \) is the number of solutions. \( B \) and \( T \) are the number of boxes and triangles. \( \text{tri-box} \) generates no boxes \( (B = 0) \) in all 3 instances. \( \text{gen} \) \( \text{prop} \) (for box and \( \text{tri-box} \)) are the generation time. \( \text{rel} \) and \( \text{prop} \) (for box and \( \text{tri-box} \)) are the time they spend on the propagation test \( M = 5000 \) and \( W = 30 \). \( \text{tri-box} \) is the fastest because it compactly represents the non-linear constraints with 1 or 2 triangles. \( \text{box} \), although faster than \( \text{rel} \), it takes a long time to generate because every box covers only a few solutions, and many boxes are needed.

<table>
<thead>
<tr>
<th>( N )</th>
<th>( \text{rel} )</th>
<th>( \text{box} )</th>
<th>( \text{tri-box} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5601</td>
<td>33.78</td>
<td>87</td>
<td>19.17</td>
</tr>
<tr>
<td>7187</td>
<td>23.58</td>
<td>57</td>
<td>20.97</td>
</tr>
<tr>
<td>2050</td>
<td>11.59</td>
<td>40</td>
<td>3.43</td>
</tr>
</tbody>
</table>

Table 1: Performance comparisons on non-linear constraints

\[ \text{5 Conclusion} \]

We have proposed a new language-independent representation, box constraint collection, for adhoc constraints. With constructive disjunction, our new representation achieves generalized arc consistency, if all constraints inside the collection do.

Future work includes improving the current greedy BCC generation algorithm, and optimizing the indexicals of a box constraint collection.

\[ \text{References} \]


