Seminar on Constraint Programming

Talk 3

Yao Li

Department of Information and Computer Science
Helsinki University of Technology

Februrary, 2009

Outline

Constraint Network

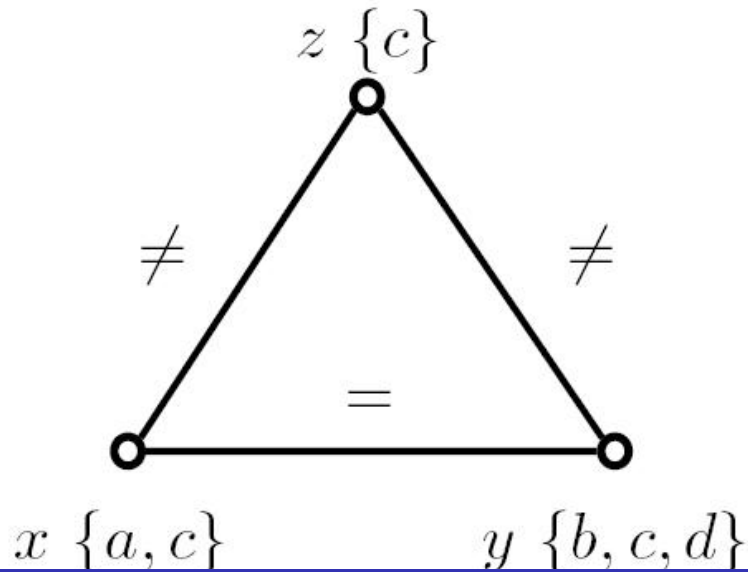
Definition

A constraint network $N(X, D, C)$ is composed of

- a finite sequence of variables $X = \{x_1, \ldots, x_n\}$
- a domain D for all the variables in X
- a set of contraints $C = \{c_1, \ldots, c_n\}$

Example

Consider a constraint network:
$\{x, y, z\}$
$\langle x \in \{a, c\}, y \in \{b, c, d\}, z = c \rangle$
$\{x = y, y \neq z, x \neq z\}$



Motivation of Constraint Propagation

- The constraint satisfaction problem (CSP) is always NP-complete
- Exploring the whole space of instantiations of a given CSP is usually too expensive
- The idea behind constraint propagation is to make the searching space small
- That is, to make the constraint network of the given CSP tight

Implementation of Constraint Propagation

- We define and achieve all kinds of local consistencies by tightening the constraint network
- After achieving a certain local consistency, the domain of the CSP is tightened
- Then the solution of the given CSP can be find by looking for global consistency in the tightened network
- We say the constraint network $N(X, D, C)$ is consistent if there exists a solution

Different Methods of Constraint Propagation

Different local consistencies with different time complexities

- arc consistency AC3 on binary network $O(ed^3)$
- arc consistency AC4 AC6 AC2001 on binary network $O(ed^2)$
- stronger than AC: path consistency, k-consistency, triangle-based consistency, neighborhood inverse consistency...
- weaker than AC: directional arc consistency, forward checking consistency, bound consistency

Outline

Basis

- GAC runs in $O(er^3d^{r+1})$
- The time complexity is extremely large with respect to large r and d
- Directional arc consistency (DAC) is simpler to enforce than AC.

Definition

A binary network $N(X, D, C)$ is DAC according to ordering $o = (x_{k_1}, \ldots, x_{k_n})$ on X, where $(k_1, \ldots, k_n)$ is a permutation of $(1, \ldots, n)$, iff for all $c(x_i, x_j) \in C$, if $x_i <_o x_j$ then x_i is arc consistent on $c(x_i, x_j)$

Example 1 of DAC

Example

$X = \{x_1, x_2, x_3\}, D(x_1) = D(x_2) = \{1..5\}, D(x_3) = \{1..3\}$
$C = \{x_1 < x_2, x_2 = x_3, x_1 > x_3\}$
What will happen after enforcing the DAC?
What will happen after enforcing the AC?

Example 2 of DAC

Example

Consider the constraint network $\langle x < y; x \in [2..10], y \in [3..7] \rangle$
What will happen after enforcing the DAC?
What will happen after enforcing the AC?

Outline

Basis

Definition

A network $N(X, D, C)$ is forward checking consistent according to the instantiation I on $Y, Y \subseteq X$, iff I is locally consistent and for all $x_i \in Y$, for all $x_j \in X \backslash Y$, for all $c(x_i, x_j) \in C$, x_j is arc consistent on $c(x_i, x_j)$.

Note

The difference between forward checking consistency and arc consistency is that the former only checks a single unassigned variable at time for consistency, while the second also checks pairs of unassigned variables for mutual consistency

Example

Example

$X = \{x_1, x_2, x_3\}, D(x_1) = D(x_2) = \{1..5\}, D(x_3) = \{1..3\}$
$C = \{x_1 < x_2, x_2 = x_3, x_1 > x_3\}$
What will happen after enforcing the FCC?
What will happen after enforcing the AC?

Outline

Bound support

Definition

Given a network N, given a constraint c, a bound support τ on c is a tuple that satisfies c and such that for all $x_i \in X(c)$, $min_D(x_i) \leq \tau[x_i] \leq max_D(x_i)$.

Example

$D(x_1) = D(X_2) = \{1, 2\}, D(x_3) = D(X_4) = \{2, 3, 5, 6\}, D(x_5) = \{5\}, D(x_6) = \{3, \ldots, 7\}$
$C = alldifferent(x_1, ..., x_6)$
Amazingly, tuple $(1, 2, 4, 6, 5, 3)$ is a bound support on C

Three kinds of bound consistency

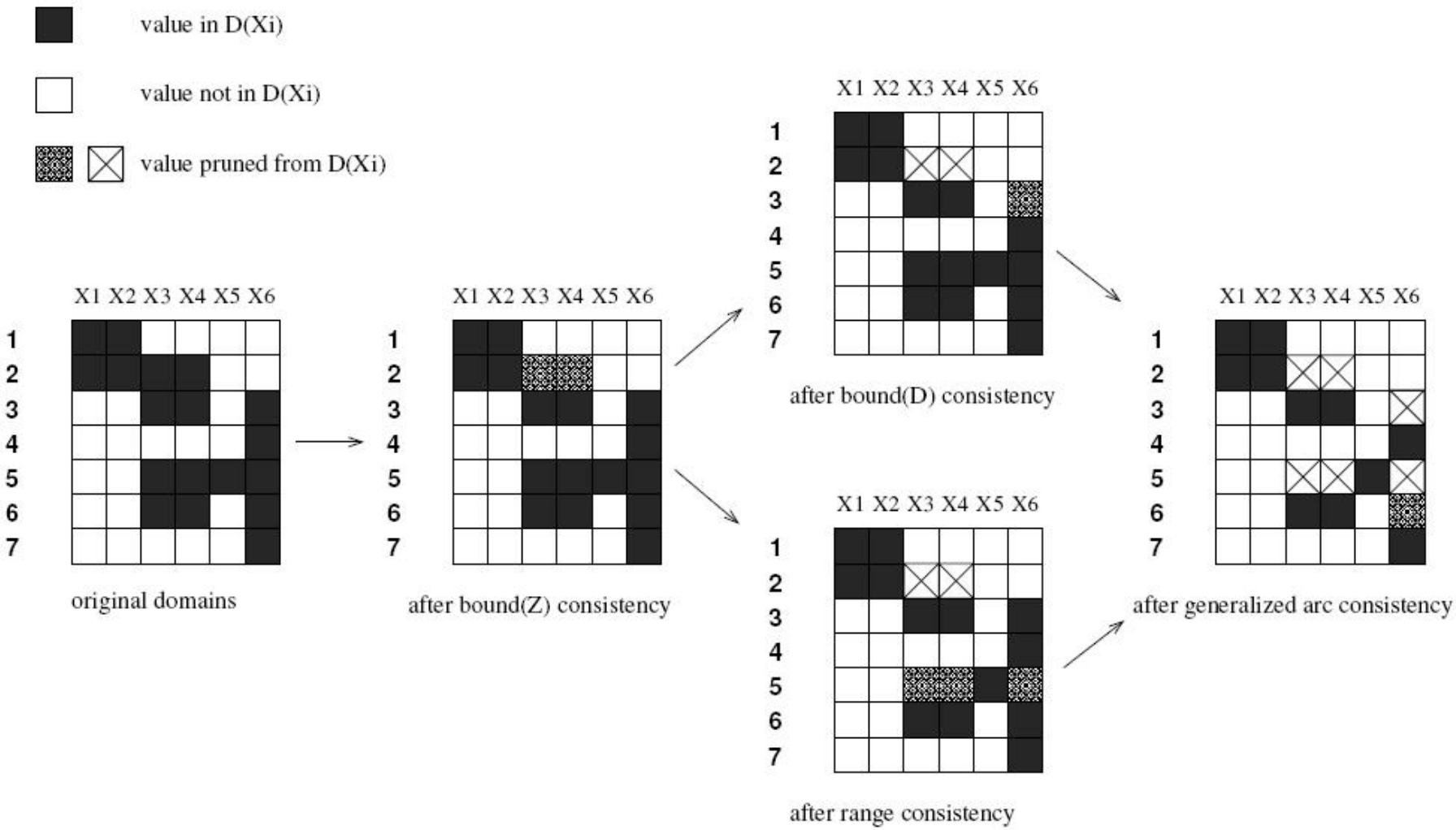
Definition

- A constraint c is bound(Z) consistent iff for all $x_i \in X(c)$, $(x_i, min_D(X_i))$ and $(x_i, max_D(X_i))$ belong to a bound support on c.
- A constraint c is range consistent iff for all $x_i \in X(c)$, for all $v_i \in D(x_i)$, (x_i, v_i) belongs to a bound support on c.
- A constraint c is bound(D) consistent iff for all $x_i \in X(c)$, $(x_i, min_D(X_i))$ and $(x_i, max_D(X_i))$ belong to a support on c.

Example

$D(x_1) = D(X_2) = \{1, 2\}, D(x_3) = D(X_4) = \{2, 3, 5, 6\}, D(x_5) = \{5\}, D(x_6) = \{3, \ldots, 7\}$
$C = alldifferent(x_1, ..., x_6)$

Example


value in D(Xi)
value not in D(Xi)
value pruned from D(Xi)
X1 X2 X3 X4 X5 X6
1 2 3 4 5 6 7
original domains
after bound(Z) consistency
after bound(D) consistency
after range consistency
after generalized arc consistency

Outline

Constraint Propagator Design

Design Principle

Making the computational load as small as possible according to spefic feature of the constraint network.

- All constraint solvers attach a specific propagation algorithm (e.g. AC3) to the specific types of contraints (e.g. arc consistency)they contain
- However, designing a constraint propagator is quite a flexible task
- More attention is paid on designing a propagator based on arithmetic constraints

Arithmetic Propagator Design based on AC3

We define the following 4 types of events

- RemValue: when a value v is removed from $D(x_i)$
- IncMin: when the minimum value of $D(x_i)$ increases
- DecMax: when the maximum value of $D(x_i)$ decreases
- Instantiate: when $D(x_i)$ becomes a singleton

Then the AC3 can be adapted like this:

Algorithm 3.10: AC3-like constraint propagation schema

```
function Constraint-Propag(in X: set): Boolean ;
  begin
1     foreach c ∈ C do perform init-propag on c and update Q with relevant events;
2     while Q ≠ ∅ do
3         select and remove (x_i, c, x_j, Mtype) from Q;
4         if Revise(x_i, c, (x_j, Mtype), Changes) then
5             if D(x_i) = ∅ then return false ;
6             foreach c' ∈ Γ^C(x_i), Mtype ∈ Changes do
7                 foreach x_j ∈ X(c'), j ≠ i do Q ← Q ∪ {(x_j, c', x_i, Mtype)};
8     return true ;
  end
  /* Γ^C(x_i) is the set of constraints with x_i in their scheme */
```

Copyright by C.Bessiere

Example

Example

Let $x_1 \leq x_2$, with $D(x_1) = D(x_2) = \{1 \ldots 100\}$
What will happen if we remove value 100 from $D(x_2)$?
What will happen if we remove 50 from $D(x_2)$?

function `revise`(**inout** x_i; **in** $c \equiv x_{k_1} \leq x_{k_2}$; **in** $(x_j, Mtype)$; **out** $Changes$): **Boolean** ;
  $Changes \leftarrow \emptyset$;
  switch $Mtype$ **do**
    case *RemValue*
      nothing;
    case *IncMin*
      if $j = k_1$ **then** remove all $v < min_D(x_j)$ from $D(x_i)$;
    case *DecMax*
      if $j = k_2$ **then** remove all $v > max_D(x_j)$ from $D(x_i)$;
    case *Instantiate*
      if $j = k_1$ **then** remove all $v < min_D(x_j)$ from $D(x_i)$;
      else remove all $v > max_D(x_j)$ from $D(x_i)$;
  $Changes \leftarrow$ the types of changes performed on $D(x_i)$;

Copyright by C.Bessiere

Outline

Motivation

- In general, the constraint propagation procedure replaces a given CSP by a simpler one, yet equivalent
- We discussed several techniques that made the domain of the network smaller
- The constraints can also become smaller using suitable methods

Example

Given a set of constraints $\{x < y, y < x\}$
After examing the constraints, we can find there is obviously no solution
This examination on constraints save us time

Classes of Specific Constraint: Global Constraint

Global constraint represents a class of constraints defined on it

Example

some global constraints

- $alldifferent(x_1, x_2, \ldots, x_n)$ is the class of constraints that are defined on any sequence of n variables such that $x_i \neq x_j$ for all $i \neq j$
- $NValue(y, [x_1, x_2, \ldots, x_n])$ is the class of constraints that are defined on any sequence of $n+1$ variables,$n \geq 1$, such that $|\{x_i | 1 \leq i \leq n\}| = y$

Important Conclusions from Relevant Literature

- Some global constraints admit a decomposition that preserves GAC
- Sometimes even decomposing a global constraint is difficult.

Summary of this chapter

- We discussed different local consistencies and their features
- Different local consistencies are enforced by different propagation algorithms
- How to tighten the given domain and how to decompose the given constraint depend on specific problem