Lecture 2: Generalized Arc Consistency

Intelligent Systems—Problem Solving
Pierre Talbot

Goals

- ⋆ Understanding the concept of consistency, in particular *generalized arc consistency*.
- ⋆ Study a generic inference algorithm.

1 Preliminaries

Let d and d' be two domain functions over a set of variables X. We define a partial order $\leq$ on domain functions as $\forall d, d' \in \mathbf{D},\ d \leq d' \Leftrightarrow \forall x \in X,\ d(x) \subseteq d'(x)$.

Exercise 1 – Partial order on $\mathbb{D}$

Let $X = \{x, y\}$ be a set of variables.

- $\{x \mapsto \{1,2\}, y \mapsto \{1,2\}\} \leq \{x \mapsto \{1,2,3\}, y \mapsto \{1,2,3\}\}$?
- $\{x \mapsto \{1,2,3\}, y \mapsto \{1,2,3\}\} \leq \{x \mapsto \{1,2\}, y \mapsto \{1,2\}\}$?
- $\{x \mapsto \{1,2\}, y \mapsto \{1,2\}\} \leq \{x \mapsto \{2,3\}, y \mapsto \{1,2\}\}$?
- $\{x \mapsto \{2,3\}, y \mapsto \{1,2\}\} \leq \{x \mapsto \{1,2\}, y \mapsto \{1,2\}\}$?
- For any $d \in \mathbf{D}$, $d \leq \{x \mapsto \{\}, y \mapsto \{\}\}$?
- For any $d, d' \in \mathbf{D}$, $d \leq d' \vee d' \leq d$?

end of exercise.

Exercise 2 – Lemma

Prove $\forall C \in \mathbf{C},\ \forall d, d' \in \mathbf{D},\ d \leq d' \Leftrightarrow sol(d, C) \subseteq sol(d', C)$.
Recall/hint:

- Cartesian product: $\{1,2\} \times \{3,4\} = \{(1,3), (1,4), (2,3), (2,4)\}$.
- Pointwise set inclusion: Let A, B, A', B' be sets. We have $A \times B \subseteq A' \times B' \Leftrightarrow A \subseteq A' \wedge B \subseteq B'$.

Exercise 3
Let $X = \{x, y, z\}$ be a set of variables, $d = \{x \mapsto \{0,1,2\}, y \mapsto \{0,1,2\}, z \mapsto \{0,1,2\}\}$ be a domain function, and $\langle d, \{x > y + z\}\rangle$ be a constraint network.

- Find a domain function d' such that $sol(d', \{x > y + z\}) = sol(d, \{x > y + z\})$ and $d' \leq d$.

- Check whether it is the smallest domain function you can find, i.e.,
  $\forall d'' \in \mathbf{D}, sol(d'', \{x > y + z\}) = sol(d, \{x > y + z\}) \Rightarrow d' \leq d''$.

end of exercise.

2 Generalized Arc Consistency

A *consistency* is a property ϕ on a constraint network $\langle d, C\rangle$. We say $\langle d, C\rangle$ is ϕ-consistent whenever $\phi(\langle d, C\rangle)$ holds. Enforcing a consistency on a constraint network can remove values from the domains of the variables and make the constraint network more explicit. It accelerates the search for a solution, as we will avoid to enumerate some locally inconsistent assignments.

An assignment in $sol(d, \{c\})$ is called a *support* of the constraint c. A *v-value* is a pair $(x, v) \in X \times \mathbb{Z}$. There exists a support for a v-value (x, v) on c iff $\exists asn \in sol(d, \{c\}),\ asn(x) = v$.

Exercise 4 – Support
Let $\langle\{x \mapsto \{0,1\}, y \mapsto \{1,2\}, z \mapsto \{0,1\}, \{x = y, x \neq z, y \neq z\}\}\rangle$ be a constraint network.

- List the supports of $x = y$:

- List the supports of $x \neq z$:

- List the supports of $y \neq z$:

end of exercise.

Generalized arc consistency (GAC) consists in removing all v-values $(x, v), v \in d(x)$ that have no support in the constraints in which x occurs. Formally, a constraint $c \in C$ is GAC-consistent iff

$$\forall x \in scp(c),\ \forall v \in d(x),\ \exists asn \in sol(d, \{c\}),\ asn(x) = v$$

A constraint network $\langle d, C\rangle$ is GAC-consistent iff every constraint $c \in C$ is GAC-consistent.

Exercise 5 – GAC-consistent
Verify whether the constraint network defined in the previous exercise is GAC-consistent. If not, list all v-values that make the constraint network not GAC-consistent.

end of exercise.

In the literature, you can sometimes read about *node consistency* which holds whenever all unary constraints are GAC-consistent, and *arc consistency* which holds whenever all binary constraints are GAC-consistent. GAC-consistency is usually the one enforced on arithmetic constraints in modern constraint solvers. It is also the strongest consistency that can be defined when considering the constraints independently.

3 Constraint Propagation

In order to enforce GAC-consistency on a constraint network, we must first do it on a single constraint.

function $revise(d, c)$
    $E = \{\}$
    for $x \in scp(c)$ **do** — *The two loops iterate over all v-values of c.*
        for $v \in d(x)$ **do**
            if $\forall a \in sol(d, \{c\}),\ a(x) \neq v$ **then** — *Check if (x, v) has no support on c.*
                $d(x) = d(x) \setminus \{v\}$ — *Remove v from the domain of x.*
                $E = E \cup \{x\}$
            end if
        end for
    end for
    return E — *Notify the caller which variables were reduced.*
end function

Exercise 6 – Revise
Let $d = \{x \mapsto \{0,1\}, y \mapsto \{1,2\}, z \mapsto \{0,1\}$ be a domain function and $\langle d, \{x = y, x \neq z, y \neq z\}\}\rangle$ be a constraint network. What is the domain after applying $revise(d, x = y)$?

end of exercise.

The simplest algorithm enforcing GAC-consistency consists in revising all the constraints until no domain is changing anymore.

function $GAC_1(d, C)$
  $b =$ `true`
  while b **do**
    $b =$ `false`
    for $c \in C$ **do**
      if $revise(d, c) \neq \{\}$ **then**
        $b =$ `true`
      end if
    end for
  end while
end function

It is however not very efficient since revising a constraint is only useful if the domain of one of its variables has changed. Since constraints are sharing variables, revising a constraint might modify a variable shared by another constraint which, in turn, should be revised. This mechanism of propagating the results of local inferences from constraints to constraints is called *constraint propagation*.

Exercise 7 – Constraint-oriented propagation scheme

Propose a function $propagate(d, C, E)$ where $\langle d, C\rangle$ is a constraint network and $E \subseteq X$ is the set of variables that changed since the last call to $propagate$. Initially, the algorithm is called with $propagate(d, C, X)$. The main idea is to maintain a queue of constraints to revise. Initially, the queue contains all the constraints with a variable in E. Let $E' = revise(d, c)$, then we must add in the queue all the constraints that have a variable in E' in their scopes. The function returns `false` if the constraint network is detected unsatisfiable, otherwise it returns `true`.

end of exercise.

Note that the algorithm $propagate$ is independent of the underlying consistency achieved. Indeed, we can switch $revise$ with a function enforcing a different consistency.