Basics of AI and Machine Learning

Constraint Satisfaction Problems: Arc Consistency

Daniel Gnad

Linköping University

Slides modified from Basel AI group, with permission

Inference

Inference

Inference

Derive additional constraints (here: unary or binary)
that are implied by the given constraints,
i.e., that are satisfied in all solutions.

example: constraint network with variables v_1, v_2, v_3
with domain $\{1, 2, 3\}$ and constraints $v_1 < v_2$ and $v_2 < v_3$.

it follows:

- v_2 cannot be equal to 3
  (new unary constraint = tighter domain of v_2)
- $R_{v_1 v_2} = \{\langle 1, 2\rangle, \langle 1, 3\rangle, \langle 2, 3\rangle\}$ can be tightened to $\{\langle 1, 2\rangle\}$
  (tighter binary constraint)
- $v_1 < v_3$
  ("new" binary constraint = trivial constraint tightened)

Trade-Off Search vs. Inference

Inference formally

For a given constraint network $\mathcal{C}$, replace $\mathcal{C}$ with an equivalent, but tighter constraint network.

Trade-off:

- the more complex the inference, and
- the more often inference is applied,
- the smaller the resulting state space, but
- the higher the complexity per search node.

When to Apply Inference?

different possibilities to apply inference:

- once as preprocessing before search
- combined with search: before recursive calls during backtracking procedure
  - already assigned variable $v \mapsto d$ corresponds to $\text{dom}(v) = \{d\}$ $\rightsquigarrow$ more inferences possible
  - during backtracking, derived constraints have to be retracted because they were based on the given assignment

$\rightsquigarrow$ powerful, but possibly expensive

Backtracking with Inference

function BacktrackingWithInference($\mathcal{C}, \alpha$):

if α is inconsistent with $\mathcal{C}$:
    return inconsistent

if α is a total assignment:
    return α

$\mathcal{C}' := \langle V, \mathrm{dom}', (R'_{uv}) \rangle :=$ copy of $\mathcal{C}$
apply inference to $\mathcal{C}'$

if $\mathrm{dom}'(v) \neq \emptyset$ for all variables v:

    select some variable v for which α is not defined

    for each $d \in$ copy of $\mathrm{dom}'(v)$ in some order:
        $\alpha' := \alpha \cup \{v \mapsto d\}$
        $\mathrm{dom}'(v) := \{d\}$
        $\alpha'' :=$ BacktrackingWithInference($\mathcal{C}', \alpha'$)
        if $\alpha'' \neq$ **inconsistent**:
            return α''

return inconsistent

Backtracking with Inference

function BacktrackingWithInference($\mathcal{C}, \alpha$):

if α is inconsistent with $\mathcal{C}$:
    return inconsistent

if α is a total assignment:
    return α

$\mathcal{C}' := \langle V, \mathrm{dom}', (R'_{uv}) \rangle :=$ copy of $\mathcal{C}$
apply inference to $\mathcal{C}'$

if $\mathrm{dom}'(v) \neq \emptyset$ for all variables v:

    select some variable v for which α is not defined

    for each $d \in$ copy of $\mathrm{dom}'(v)$ in some order:
        $\alpha' := \alpha \cup \{v \mapsto d\}$
        $\mathrm{dom}'(v) := \{d\}$
        $\alpha'' :=$ BacktrackingWithInference($\mathcal{C}', \alpha'$)
        if $\alpha'' \neq$ **inconsistent**:
            return α''

return inconsistent

Backtracking with Inference: Discussion

- Inference is a placeholder:
  different inference methods can be applied.
- Inference methods can recognize unsolvability (given α)
  and indicate this by clearing the domain of a variable.
- Efficient implementations of inference are often incremental:
  the last assigned variable/value pair $v \mapsto d$ is taken
  into account to speed up the inference computation.

Forward Checking

Forward Checking

We start with a simple inference method:

Forward Checking

Let α be a partial assignment.
Inference: For all unassigned variables v in α,
remove all values from the domain of v that are in conflict
with already assigned variable/value pairs in α.

$\rightsquigarrow$ definition of conflict as in the previous chapter

Incremental computation:

- When adding $v \mapsto d$ to the assignment,
  delete all pairs that conflict with $v \mapsto d$.

Forward Checking: Discussion

properties of forward checking:

- correct inference method (retains equivalence)
- affects domains (= unary constraints), but not binary constraints
- consistency check at the beginning of the backtracking procedure no longer needed (Why?)
- cheap, but often still useful inference method

$\rightsquigarrow$ apply at least forward checking in the backtracking procedure

In the following, we will consider more powerful inference methods.

Arc Consistency

Arc Consistency: Definition

Definition (Arc Consistent)

Let $\mathcal{C} = \langle V, \mathrm{dom}, (R_{uv}) \rangle$ be a constraint network.

(a) The variable $v \in V$ is arc consistent
with respect to another variable $v' \in V$,
if for every value $d \in \mathrm{dom}(v)$
there exists a value $d' \in \mathrm{dom}(v')$ with $\langle d, d' \rangle \in R_{vv'}$.

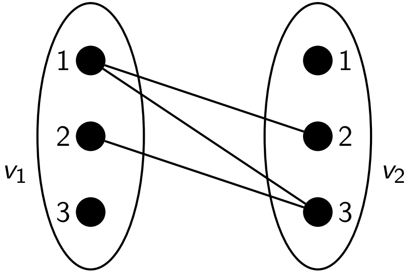
(b) The constraint network $\mathcal{C}$ is arc consistent,
if every variable $v \in V$ is arc consistent
with respect to every other variable $v' \in V$.

remarks:

- definition for variable pair is not symmetrical
- v always arc consistent with respect to v'
  if the constraint between v and v' is trivial

Arc Consistency: Example

Consider a constraint network with variables v_1 and v_2,
domains $\mathrm{dom}(v_1) = \mathrm{dom}(v_2) = \{1, 2, 3\}$
and the constraint expressed by $v_1 < v_2$.



Arc consistency of v_1 with respect to v_2
and of v_2 with respect to v_1 are violated.

Enforcing Arc Consistency

- Enforcing arc consistency, i.e., removing values from $\mathrm{dom}(v)$ that violate the arc consistency of v with respect to v', is a correct inference method. (Why?)
- more powerful than forward checking (Why?)

Enforcing Arc Consistency

- Enforcing arc consistency, i.e., removing values from $\text{dom}(v)$ that violate the arc consistency of v with respect to v', is a correct inference method. (Why?)
- more powerful than forward checking (Why?)
  - ⇝ Forward checking is a special case: enforcing arc consistency of all variables with respect to the just assigned variable corresponds to forward checking.

We will next consider an algorithm that enforces arc consistency.

Processing Variable Pairs: revise

function revise($\mathcal{C}, v, v'$):

$\langle V, \text{dom}, (R_{uv}) \rangle := \mathcal{C}$

for each $d \in \text{dom}(v)$:

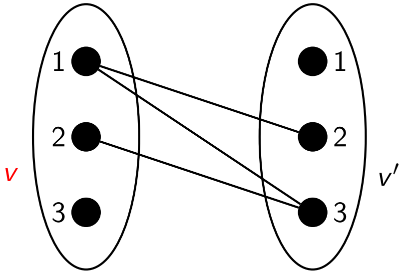
  if there is no $d' \in \text{dom}(v')$ with $\langle d, d' \rangle \in R_{vv'}$:

    remove d from $\text{dom}(v)$

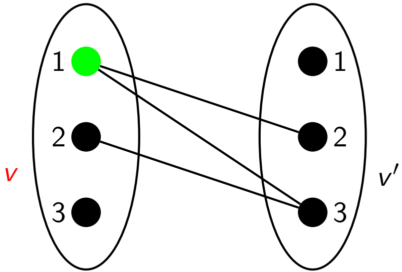
input: constraint network $\mathcal{C}$ and two variables v, v' of $\mathcal{C}$

effect: v arc consistent with respect to v'.
All violating values in $\text{dom}(v)$ are removed.

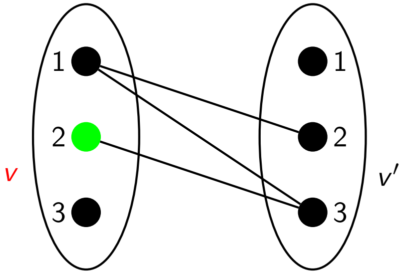
Example: revise



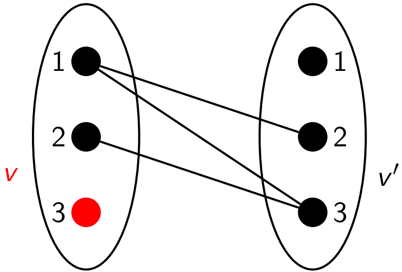
Example: revise



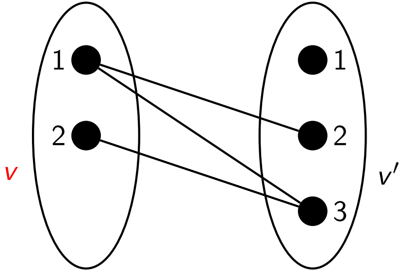
Example: revise



Example: revise



Example: revise



Enforcing Arc Consistency: AC-1

function AC-1($\mathcal{C}$):

$\langle V, \text{dom}, (R_{uv}) \rangle := \mathcal{C}$

repeat

    for each nontrivial constraint R_{uv}:

        revise($\mathcal{C}, u, v$)

        revise($\mathcal{C}, v, u$)

until no domain has changed in this iteration

input: constraint network $\mathcal{C}$

effect: transforms $\mathcal{C}$ into equivalent arc consistent network

AC-1: Discussion

- AC-1 does the job, but is rather inefficient.
- Drawback: Variable pairs are often checked again and again although their domains have remained unchanged.
- These (redundant) checks can be saved.

$\rightsquigarrow$ more efficient algorithm: AC-3, not covered here

Summary

Summary: Inference

- inference: derivation of additional constraints that are implied by the known constraints

$\rightsquigarrow$ tighter equivalent constraint network

- trade-off search vs. inference
- inference as preprocessing or integrated into backtracking

Summary: Forward Checking, Arc Consistency

- cheap and easy inference: forward checking
  - remove values that conflict with already assigned values
- more expensive and more powerful: arc consistency
  - iteratively remove values without a suitable "partner value" for another variable until fixed-point reached