Seminar: Search and Optimization

Consistency-Enforcing and Constraint Propagation

Presentation: Martin Wehrle

University of Basel

October 16, 2014

Inference

Inference

What is Inference?

- Derivation of additional constraints that are implied from known constraints

Why can Inference be Useful?

- Narrows the search space of possible partial solutions
- Search for solutions becomes more focused

Inference

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 = restriction of the domain of v_2)
- $R_{v_1 v_2} = \{\langle 1, 2\rangle, \langle 1, 3\rangle, \langle 2, 3\rangle\}$ can be made stronger to $\{\langle 1, 2\rangle\}$
  (tightened binary constraint)

Inference

Inference formally

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

Trade-off:

- the more complex the interference,
- the more additional constraints can be inferred, but
- the higher the time complexity

Inference

In this talk, we consider increasingly powerful inference methods.

Outline

- Arc-consistency
- Path-consistency
- *i*-consistency

Arc-Consistency

Arc-Consistency: Definition

Definition (Arc-Consistent)

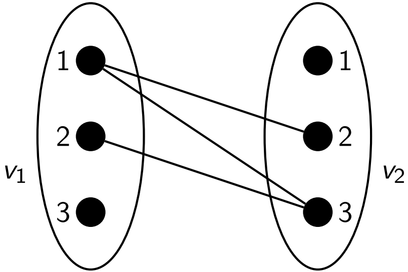
Let $\mathcal{R} = (X, D, C)$ be a constraint network.

(a) A variable $v \in X$ is arc-consistent with respect to another variable $v' \in X$ if for every value $d \in D_v$ there is a value $d' \in D_{v'}$ such that $\langle d, d' \rangle \in R_{vv'}$.

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

Arc-Consistency: Example

Consider a constraint network with variables v_1 and v_2, domains $D_{v_1} = D_{v_2} = \{1, 2, 3\}$, and the constraint $v_1 < v_2$.



- v_1 not arc-consistent with respect to v_2
- v_2 not arc-consistent with respect to v_1

Enforcing Arc-Consistency

- Enforcing arc-consistency, i. e., removing values of D_v that violate the arc-consistency of v with respect to v' is a correct inference method. (Why?)
- In the following, we consider algorithms to enforce arc-consistency.

Single Pair of Variables: Function revise

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

$(X, D, C) := \mathcal{R}$

for each $d \in D_v$:

  if there is no $d' \in D_{v'}$ with $\langle d, d' \rangle \in R_{vv'}$:

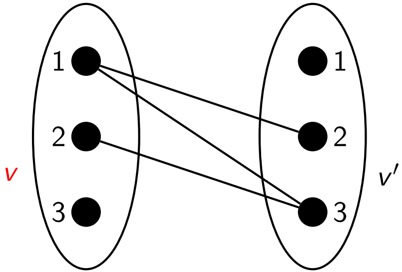
    remove d from D_v

Input: Constraint network $\mathcal{R}$ and two variables v, v' in $\mathcal{R}$

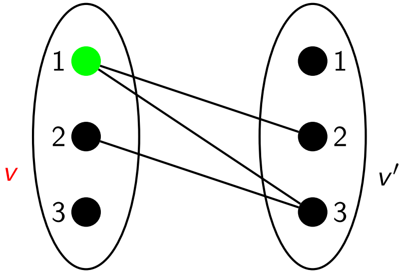
Effect: Enforces arc-consistency of v with respect to v'.
All violating values are removed from D_v.

Time complexity: $O(k^2)$, where k bounds the domain size.

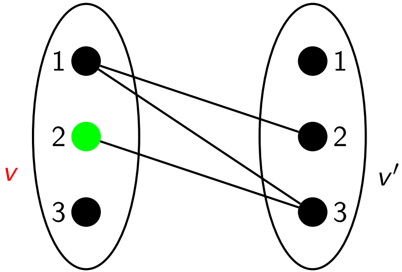
Example: revise



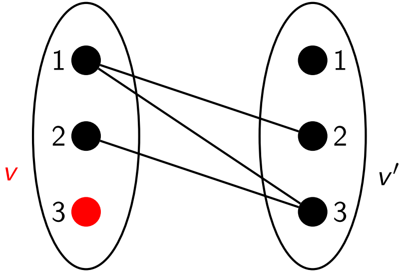
Example: revise



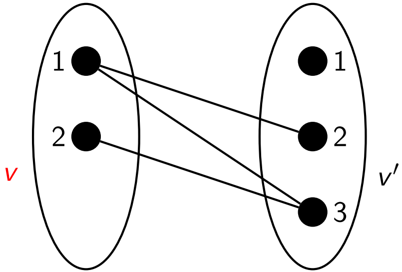
Example: revise



Example: revise



Example: revise



Enforcing Arc-Consistency: AC-1

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

$(X, D, C) := \mathcal{R}$

repeat

  for each nontrivial constraint $R_{uv} \in C$:

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

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

until no domain has changed in this iteration

Input: Constraint network $\mathcal{R}$

Effect: transforms $\mathcal{R}$ into equivalent network that is arc-consistent

Time complexity: ?

Enforcing Arc-Consistency: AC-1

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

$(X, D, C) := \mathcal{R}$

repeat

  for each nontrivial constraint $R_{uv} \in C$:

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

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

until no domain has changed in this iteration

Input: Constraint network $\mathcal{R}$

Effect: transforms $\mathcal{R}$ into equivalent network that is arc-consistent

Time complexity: $O(n \cdot e \cdot k^3)$, for n variables, e nontrivial constraints, and k maximal domain size

AC-1: Discussion

- AC-1 does the job, but in an inefficient way.
- Often variable pairs are checked again and again although their domains have not changed.
- These checks can be saved.

$\leadsto$ more efficient algorithm: AC-3

Enforcing Arc-Consistency: AC-3

Idea: store variable pairs that are potentially inconsistent in a queue

function AC-3($\mathcal{R}$):

$(X, D, C) := \mathcal{R}$

$queue := \emptyset$

for each nontrivial constraint $R_{uv} \in C$:

    insert $\langle u, v \rangle$ into $queue$

    insert $\langle v, u \rangle$ into $queue$

while $queue \neq \emptyset$:

    remove an arbitrary element $\langle u, v \rangle$ from $queue$

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

    if D_u changed in the call to revise:

        for each $w \in X \setminus \{u, v\}$ where R_{wu} is nontrivial:

            insert $\langle w, u \rangle$ into $queue$

AC-3: Discussion

- *queue* can be an arbitrary data structure that allows for "insert" and "get" operations (the order of getting the variable pairs does not affect the result)

$\rightsquigarrow$ efficient e. g. a stack

- AC-3 has the same effect as AC-1: it enforces arc-consistency
- Proof idea: Invariant of the **while** loop: If $\langle u, v \rangle \notin queue$, then u is arc-consistent with respect to v

AC-3: Time Complexity

Proposition (Time complexity of AC-3)

Let $\mathcal{R}$ be a constraint network with e nontrivial constraints and maximal domain size k.
Then AC-3 has time complexity $O(e \cdot k^3)$.

AC-3: Time Complexity (Proof)

Proof

Consider a pair $\langle u, v \rangle$ for which there is a nontrivial constraint R_{uv}. There are e such pairs.

AC-3: Time Complexity (Proof)

Proof

Consider a pair $\langle u, v \rangle$ for which there is a nontrivial constraint R_{uv}. There are e such pairs.

Every time a pair is inserted into the queue (except for the first time), the domain of the second variable has been reduced before.

AC-3: Time Complexity (Proof)

Proof

Consider a pair $\langle u, v \rangle$ for which there is a nontrivial constraint R_{uv}. There are e such pairs.

Every time a pair is inserted into the queue (except for the first time), the domain of the second variable has been reduced before.

This can happen at most k times.

AC-3: Time Complexity (Proof)

Proof

Consider a pair $\langle u, v \rangle$ for which there is a nontrivial constraint R_{uv}. There are e such pairs.

Every time a pair is inserted into the queue (except for the first time), the domain of the second variable has been reduced before.

This can happen at most k times.

Hence, every pair $\langle u, v \rangle$ is inserted into the queue at most $k + 1$ times $\rightsquigarrow$ all in all, we have at most $O(ek)$ insert operations.

AC-3: Time Complexity (Proof)

Proof

Consider a pair $\langle u, v \rangle$ for which there is a nontrivial constraint R_{uv}. There are e such pairs.

Every time a pair is inserted into the queue (except for the first time), the domain of the second variable has been reduced before.

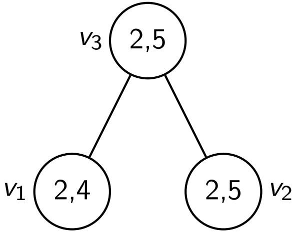
This can happen at most k times.

Hence, every pair $\langle u, v \rangle$ is inserted into the queue at most $k + 1$ times $\rightsquigarrow$ all in all, we have at most $O(ek)$ insert operations.

This restricts the number of **while** iterations to $O(ek)$, therefore the revise calls need time at most $O(ek) \cdot O(k^2) = O(ek^3)$.

AC-3: Example

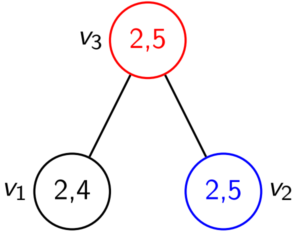
Consider the constraint network with three variables v_1, v_2, v_3 with $D_{v_1} = \{2, 4\}$ and $D_{v_2} = D_{v_3} = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides evenly").



Queue
(v_1, v_3)
(v_3, v_1)
(v_2, v_3)
(v_3, v_2)

AC-3: Example

Consider the constraint network with three variables v_1, v_2, v_3 with $D_{v_1} = \{2, 4\}$ and $D_{v_2} = D_{v_3} = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides evenly").

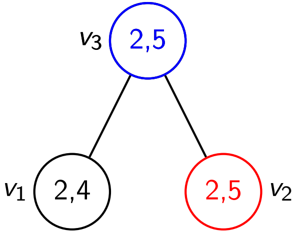


Queue

(v_1, v_3)
(v_3, v_1)
(v_2, v_3)
(v_3, v_2)

AC-3: Example

Consider the constraint network with three variables v_1, v_2, v_3 with $D_{v_1} = \{2, 4\}$ and $D_{v_2} = D_{v_3} = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides evenly").

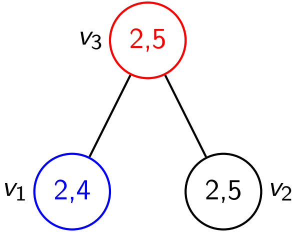


Queue

(v_1, v_3)
(v_3, v_1)
(v_2, v_3)

AC-3: Example

Consider the constraint network with three variables v_1, v_2, v_3
with $D_{v_1} = \{2, 4\}$ and $D_{v_2} = D_{v_3} = \{2, 5\}$
and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides evenly").

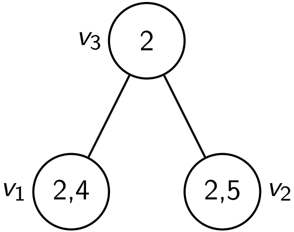


Queue

(v_1, v_3)
(v_3, v_1)

AC-3: Example

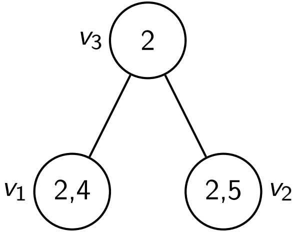
Consider the constraint network with three variables v_1, v_2, v_3 with $D_{v_1} = \{2, 4\}$ and $D_{v_2} = D_{v_3} = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides evenly").



Queue
(v_1, v_3)

AC-3: Example

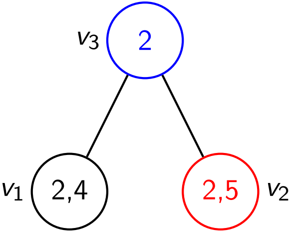
Consider the constraint network with three variables v_1, v_2, v_3 with $D_{v_1} = \{2, 4\}$ and $D_{v_2} = D_{v_3} = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides evenly").



Queue
(v_1, v_3)
(v_2, v_3)

AC-3: Example

Consider the constraint network with three variables v_1, v_2, v_3 with $D_{v_1} = \{2, 4\}$ and $D_{v_2} = D_{v_3} = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides evenly").

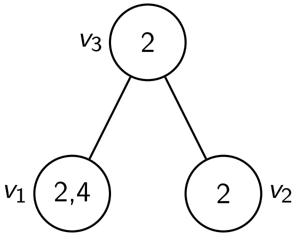


Queue

(v_1, v_3)
(v_2, v_3)

AC-3: Example

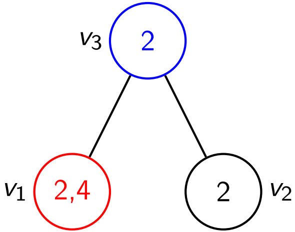
Consider the constraint network with three variables v_1, v_2, v_3 with $D_{v_1} = \{2, 4\}$ and $D_{v_2} = D_{v_3} = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides evenly").



Queue
(v_1, v_3)

AC-3: Example

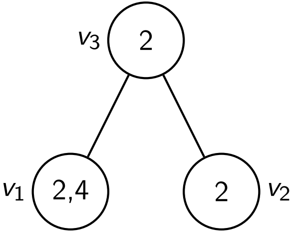
Consider the constraint network with three variables v_1, v_2, v_3 with $D_{v_1} = \{2, 4\}$ and $D_{v_2} = D_{v_3} = \{2, 5\}$ and the constraints $v_3|v_1$ and $v_3|v_2$ ("divides evenly").



Queue
(v_1, v_3)

AC-3: Example

Consider the constraint network with three variables v_1, v_2, v_3 with $D_{v_1} = \{2, 4\}$ and $D_{v_2} = D_{v_3} = \{2, 5\}$ and the constraints $v_3 | v_1$ and $v_3 | v_2$ ("divides evenly").



Queue

Path-Consistency

Beyond Arc-Consistency: Path-Consistency

Recall: Idea of Arc-Consistency:

- For every value of variable u there exists a consistent value for every other variable v
- Values of u that do not have this property are not allowed

$\rightsquigarrow$ tightens unary constraint on u

Idea can be extended to three variables (path-consistency):

- For every common valuation of u, v there must be a consistent value for every other variable w
- Pairs of values for u and v that do not have this property are not allowed

$\rightsquigarrow$ tightens binary constraint on u and v

Path-Consistency: Definition

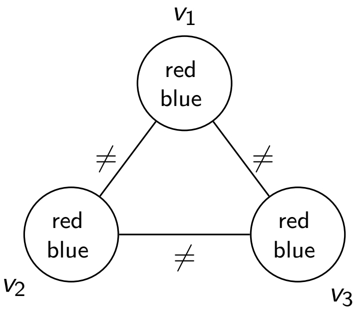
Definition (Path-Consistent)

Let $\mathcal{R} = (X, D, C)$ be a constraint network.

(a) Two different variables $u, v \in X$ are path-consistent with respect to a third variable $w \in X$ if for all values $d_u \in D_u, d_v \in D_v$ with $\langle u, v \rangle \in R_{uv}$, there is a value $d_w \in D_w$ such that $\langle d_u, d_w \rangle \in R_{uw}$ and $\langle d_v, d_w \rangle \in R_{vw}$.

(b) The constraint network $\mathcal{R}$ is path-consistent, if for three different variables u, v, w, it holds that u and v are path-consistent with respect to w.

Path-Consistency: Example



arc-consistent, but not path-consistent

Triple of Variables: revise-3

Analogous to revise for arc-consistency:

function revise-3($\mathcal{R}, u, v, w$):

$(X, D, C) := \mathcal{R}$

for each $\langle d_u, d_v \rangle \in R_{uv}$:
  if there is no $d_w \in D_w$ with
    $\langle d_u, d_w \rangle \in R_{uw}$ **and** $\langle d_v, d_w \rangle \in R_{vw}$:
      remove $\langle d_u, d_v \rangle$ from R_{uv}

Input: Constraint network $\mathcal{R}$ and three variables u, v, w of $\mathcal{R}$

Effect: Turns u, v path-consistent with respect to w.
All violating pairs of variables are removed from R_{uv}.

Time complexity: $O(k^3)$, where k is the maximal domain size

Enforcing Path-Consistency: PC-2

Analogous to AC-3 for arc-consistency:

function PC-2($\mathcal{R}$):

$(X, D, C) := \mathcal{R}$
$queue := \emptyset$
for each set of two variables $\{u, v\}$:
    for each $w \in X \setminus \{u, v\}$:
        insert $\langle u, v, w \rangle$ into *queue*
while $queue \neq \emptyset$:
    remove any element $\langle u, v, w \rangle$ from *queue*
    revise-3($\mathcal{R}, u, v, w$)
    if R_{uv} changed in the call to revise-3:
        for each $w' \in X \setminus \{u, v\}$:
            insert $\langle w', u, v \rangle$ into *queue*
            insert $\langle w', v, u \rangle$ into *queue*

Path-Consistency: Summary

- Generalization of arc-consistency (which considers pairs of variables) to path-consistency (which considers triples of variables)
- Arc-consistency tightens unary constraints
- Path-consistency tightens binary constraints

i-Consistency

i-Consistency: Idea

- Further generalize previous concepts
- For every valuation of $v_1, \dots, v_{i-1}$ there must exist a corresponding consistent valuation of every other variable v_i
- Otherwise the valuation for $v_1, \dots, v_{i-1}$ (that is not extendable to v_i) is not allowed

$\rightsquigarrow$ tightens $(i-1)$-ary constraint on $v_1, \dots, v_{i-1}$

$\rightsquigarrow$ also affects non-binary constraints

i-Consistency: Definition

Definition (i-Consistent)

Let $\mathcal{R} = (X, D, C)$ be a constraint network.

(a) A relation $R_S \in C$ with $|S| = i - 1$ is i-consistent with respect to a variable $y \notin S$ if for every $t \in R_S$, there is a value $d \in D_y$ such that t and d are consistent.

(b) The constraint network $\mathcal{R}$ is i-consistent if for any consistent valuation of any $i - 1$ distinct variables in X, there is a valuation of any ith variable such that the i values together satisfy all of the constraints among the i variables.

i-Consistency: Example

Constraint network $\mathcal{R}$ for the 4-queens problem.

	v_1	v_2	v_3	v_4
1	Q			
2				
3		Q		
4				

	v_1	v_2	v_3	v_4
1	Q			
2			Q	
3				
4		Q		

- $\mathcal{R}$ is 2-consistent: All single queen placements can be extended
- $\mathcal{R}$ is not 3-consistent: There are valid placements of 2 queens that cannot be extended (left)
- $\mathcal{R}$ is not 4-consistent: There are valid placements of 3 queens that cannot be extended (right)

Enforcing i-Consistency

- There exist extensions of arc- and path-consistency algorithms to enforce i-consistency
- We are not going into more detail here

Summary

Inference: Summary

- Inference: Derivation of additional constraints that are implied by the given constraints

$\rightsquigarrow$ equivalent but tighter constraint network

- Useful for search-based solving approaches ($\rightsquigarrow$ next chapters)
- Trade-off: Number of inferred constraints vs. time complexity