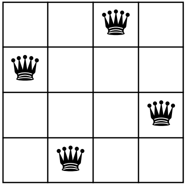
Constraint Programming

Lecture 2: consistency techniques

Master 1 informatique – Université de Nantes

Laurent Granvilliers

4-queens



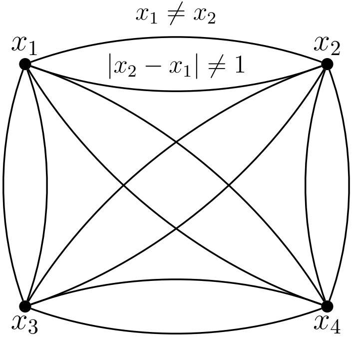
$\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ s.t. $x_i = j$ iff in row i the queen is in column j for all $i \in 1..4$.

$D_i = 1..4$ for all $i \in 1..4$.

$$
\begin{aligned}
C = \{ \quad & x_1 \neq x_2, \quad |x_2 - x_1| \neq 1, \\
& x_1 \neq x_3, \quad |x_3 - x_1| \neq 2, \\
& x_1 \neq x_4, \quad |x_4 - x_1| \neq 3, \\
& x_2 \neq x_3, \quad |x_3 - x_2| \neq 1, \\
& x_2 \neq x_4, \quad |x_4 - x_2| \neq 2, \\
& x_3 \neq x_4, \quad |x_4 - x_3| \neq 1 \quad \}
\end{aligned}
$$

Constraint network

A CSP $\langle C, X, D \rangle$ composed of binary constraints is often represented by a graph with vertices X and edges C.



Consistency and pruning

Informal definition

A consistency property is a satisfiability condition of a restriction of a CSP to a subset of variables or constraints.

Consistency and pruning

Informal definition

A consistency property is a satisfiability condition of a restriction of a CSP to a subset of variables or constraints.

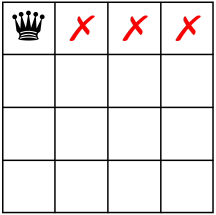
Pruning

Violating a consistency property leads to prune the domains.

- elimination of conflicts with respect to this property
- propagation of modifications through the network

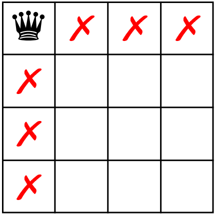
Example

Consider the 4-queens problem and assign $x_1 = 1$. Then the values $2, 3, 4$ can be removed from D_1.



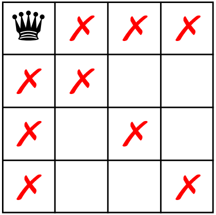
Example

Now consider the constraints $x_1 \neq x_i$ for $i = 2, 3, 4$. Then the value 1 can be removed from D_2, D_3, D_4.



Example

Now consider the constraints $|x_i - x_1| \neq i - 1$. Then the value i can be removed from D_i for $i = 2, 3, 4$.



Node consistency

Definition

Let c be a unary constraint on $X_c = \{x_i\}$ and let D_i be the domain of x_i. The variable x_i is node consistent relative to constraint c if and only if

$$\forall a_i \in D_i : \; a_i \in R_c.$$

Node consistency

Definition

Let c be a unary constraint on $X_c = \{x_i\}$ and let D_i be the domain of x_i. The variable x_i is node consistent relative to constraint c if and only if

$$\forall a_i \in D_i : \; a_i \in R_c.$$

- Given $D_i = 0..2$, x_i is not node consistent relative to $c : x_i^2 - 3x_i + 2 = 0$ since c is violated when $x_i \to 0$.

Node consistency

Definition

Let c be a unary constraint on $X_c = \{x_i\}$ and let D_i be the domain of x_i. The variable x_i is node consistent relative to constraint c if and only if

$$\forall a_i \in D_i : \; a_i \in R_c.$$

- Given $D_i = 0..2$, x_i is not node consistent relative to $c : x_i^2 - 3x_i + 2 = 0$ since c is violated when $x_i \rightarrow 0$.

  $\Longrightarrow$ 0 can be removed from D_i

Node consistency

Definition

Let c be a unary constraint on $X_c = \{x_i\}$ and let D_i be the domain of x_i. The variable x_i is node consistent relative to constraint c if and only if

$$\forall a_i \in D_i : \; a_i \in R_c.$$

- Given $D_i = 0..2$, x_i is not node consistent relative to $c : x_i^2 - 3x_i + 2 = 0$ since c is violated when $x_i \to 0$.

  $\Longrightarrow$ 0 can be removed from D_i

- Given $D_i = 1..2$, x_i is node consistent relative to c since $x_i \to 1$ and $x_i \to 2$ are two solutions $(1, 2 \in R_c)$.

Arc consistency

Definition

Let c be a binary constraint on $X_c = \{x_i, x_j\}$ and let D_i, D_j be the domains of x_i, x_j. The variable x_i is arc consistent relative to constraint c if and only if

$$\forall a_i \in D_i\ \exists a_j \in D_j :\ (a_i, a_j) \in R_c.$$

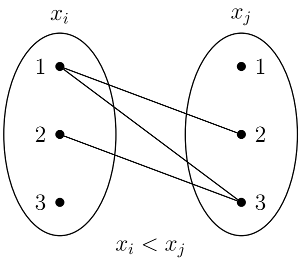
Arc consistency

Definition

Let c be a binary constraint on $X_c = \{x_i, x_j\}$ and let D_i, D_j be the domains of x_i, x_j. The variable x_i is arc consistent relative to constraint c if and only if

$$\forall a_i \in D_i \ \exists a_j \in D_j : \ (a_i, a_j) \in R_c.$$

not arc consistent

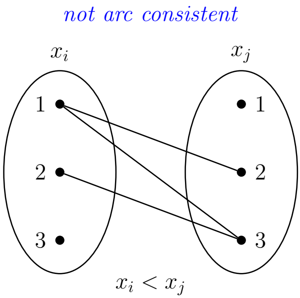


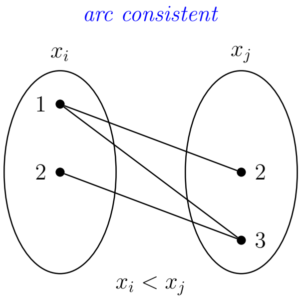
Arc consistency

Definition

Let c be a binary constraint on $X_c = \{x_i, x_j\}$ and let D_i, D_j be the domains of x_i, x_j. The variable x_i is arc consistent relative to constraint c if and only if

$$\forall a_i \in D_i \; \exists a_j \in D_j : \; (a_i, a_j) \in R_c.$$





Arc consistency (cont)

Definition

A binary CSP is arc consistent if for every constraint c and every variable $x \in X_c$, x is arc consistent relative to c.

Arc consistency (cont)

Definition

A binary CSP is arc consistent if for every constraint c and every variable $x \in X_c$, x is arc consistent relative to c.

Check the satisfiability and the arc consistency properties in the following examples. Is it possible to prune domains?

Arc consistency (cont)

Definition

A binary CSP is arc consistent if for every constraint c and every variable $x \in X_c$, x is arc consistent relative to c.

Check the satisfiability and the arc consistency properties in the following examples. Is it possible to prune domains?

1. $C = \{x_1 \neq x_2, x_2 \neq x_3, x_3 \neq x_1\},\ D_i = 1..2\ \forall i$

Arc consistency (cont)

Definition

A binary CSP is arc consistent if for every constraint c and every variable $x \in X_c$, x is arc consistent relative to c.

Check the satisfiability and the arc consistency properties in the following examples. Is it possible to prune domains?

1. $C = \{x_1 \neq x_2, x_2 \neq x_3, x_3 \neq x_1\},\ D_i = 1..2\ \forall i$
2. $C = \{x_1 < x_2, x_2 < x_3, x_3 < x_1\},\ D_i = 1..3\ \forall i$

Arc consistency (cont)

Definition

A binary CSP is arc consistent if for every constraint c and every variable $x \in X_c$, x is arc consistent relative to c.

Check the satisfiability and the arc consistency properties in the following examples. Is it possible to prune domains?

1. $C = \{x_1 \neq x_2, x_2 \neq x_3, x_3 \neq x_1\},\ D_i = 1..2\ \forall i$
2. $C = \{x_1 < x_2, x_2 < x_3, x_3 < x_1\},\ D_i = 1..3\ \forall i$
3. $C = \{x_1 < x_2, x_2 < x_3\},\ D_i = 1..3\ \forall i$

AC-based pruning

```
ReviseAC(c(x_i,x_j):constraint, D_i:domain, D_j:domain)
begin
    foreach a_i in D_i do
        if not HasSupport(c,a_i,D_j) then
            remove a_i from D_i
        endif
    endfor
end

HasSupport(c(x_i,x_j):constraint, a_i:value, D_j:domain)
    return bool
begin
    foreach a_j in D_j do
        if (a_i,a_j) ∈ R_c then return true endif
    endfor
    return false
end
```

AC-based pruning (cont)

Maximal consistency

The new domain computed by `ReviseAC`(c, D_i, D_j) is the largest domain included in D_i such that x_i is arc consistent relative to c.

AC-based pruning (cont)

Maximal consistency

The new domain computed by `ReviseAC`(c, D_i, D_j) is the largest domain included in D_i such that x_i is arc consistent relative to c.

Worst-case complexity

$O(d^2)$ membership tests where d bounds the domains size.

AC-based propagation

```
AC1(P = (C, X, D) : CSP)
var modified: bool
repeat
   modified := false
   foreach c(x_i, x_j) ∈ C do
      E_i := D_i
      E_j := D_j
      ReviseAC(c(x_i, x_j), D_i, D_j)
      ReviseAC(c(x_j, x_i), D_j, D_i)
      modified := modified or (E_i ≠ D_i) or (E_j ≠ D_j)
   endfor
until not modified
```

AC-based propagation (cont)

Maximal consistency

`AC1` returns the largest domains $D' = \{D'_1, \ldots, D'_n\}$ included in D such that the CSP (C, X, D') is arc consistent.

AC-based propagation (cont)

Maximal consistency

`AC1` returns the largest domains $D' = \{D'_1, \ldots, D'_n\}$ included in D such that the CSP (C, X, D') is arc consistent.

Worst-case complexity

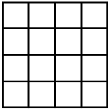
$O(nmd^3)$ tests.

Maintaining arc consistency

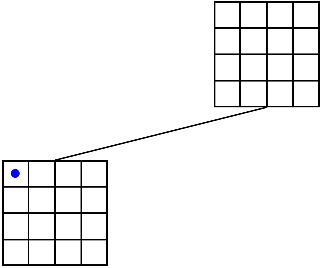
Maintaining AC

MAC is a branch-and-prune algorithm for solving finite domain CSPS such that an AC-based propagation algorithm is used as pruning method.

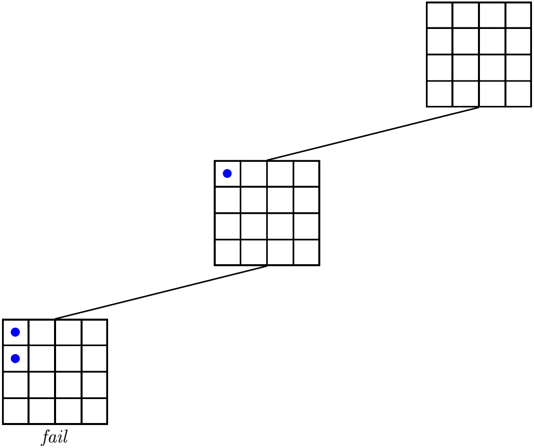
4-queens (backtracking)



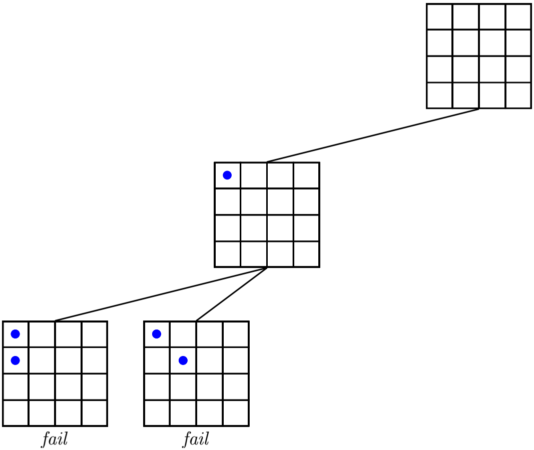
4-queens (backtracking)



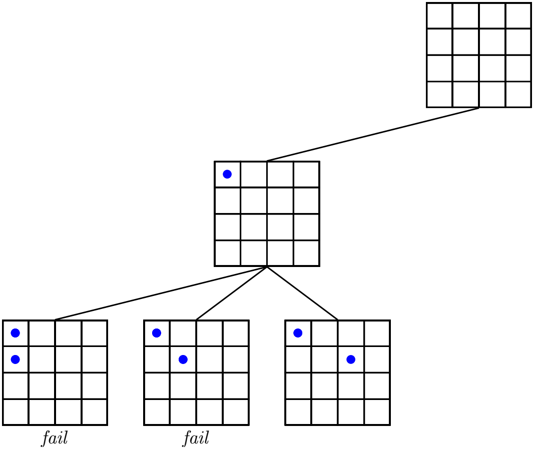
4-queens (backtracking)



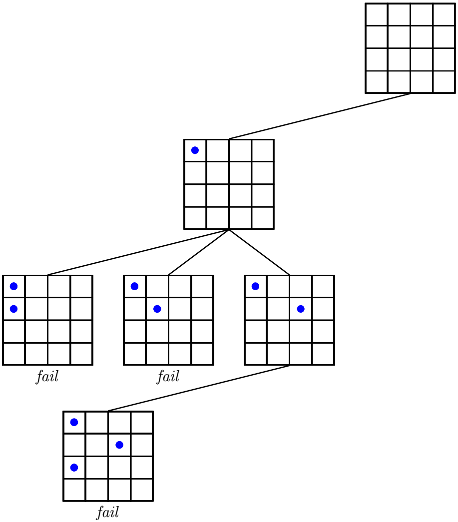
4-queens (backtracking)



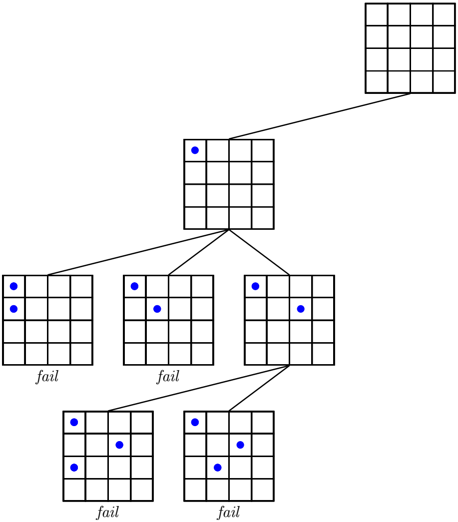
4-queens (backtracking)



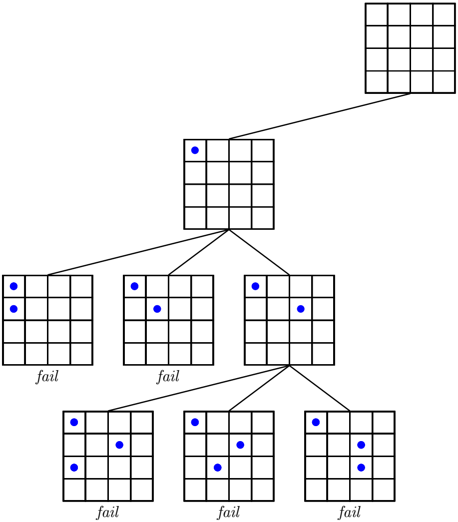
4-queens (backtracking)



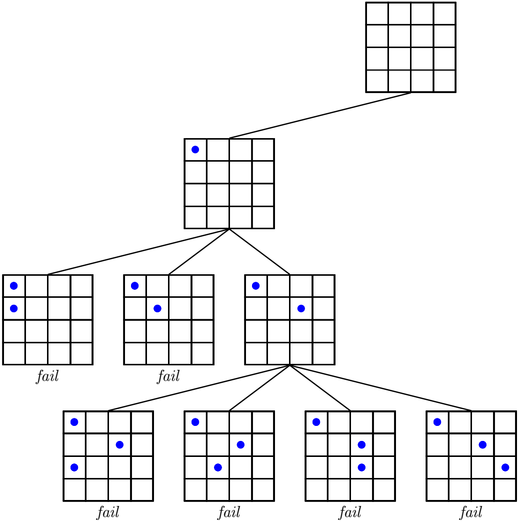
4-queens (backtracking)



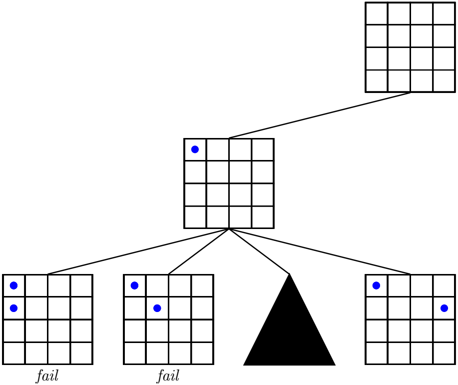
4-queens (backtracking)



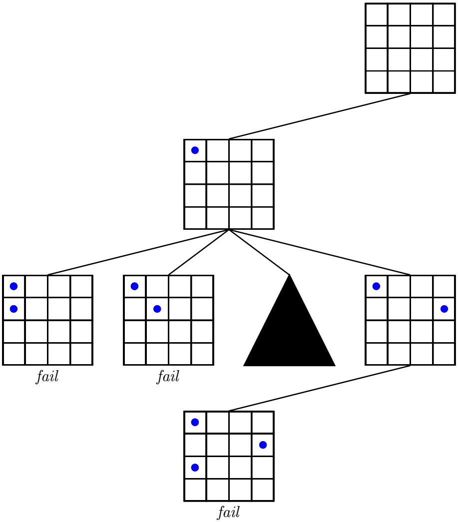
4-queens (backtracking)



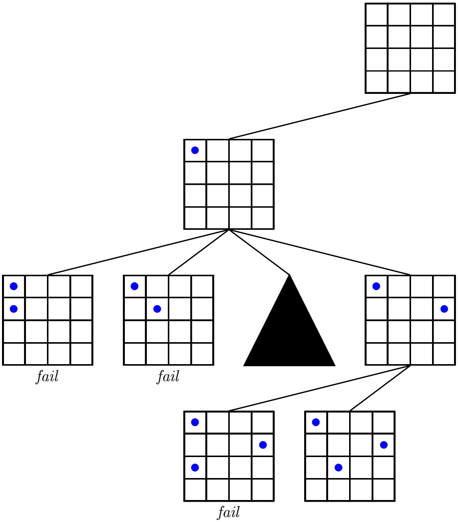
4-queens (backtracking)



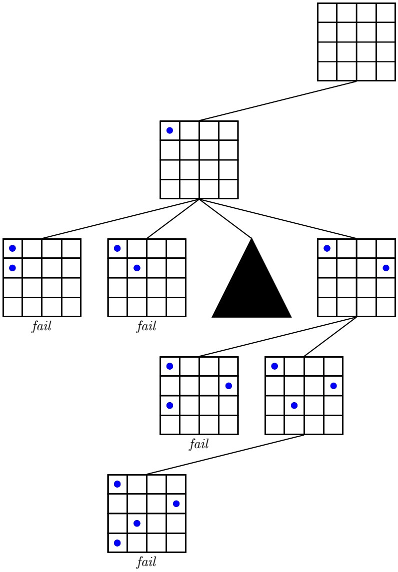
4-queens (backtracking)



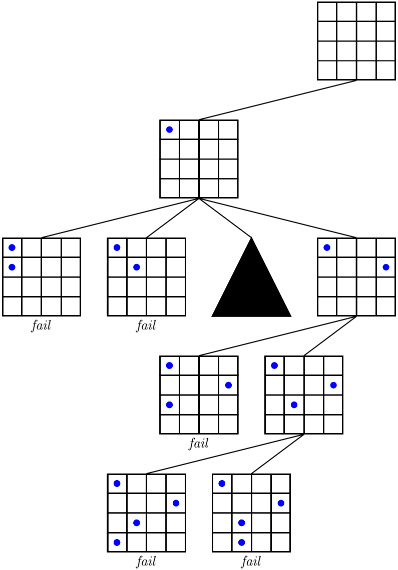
4-queens (backtracking)



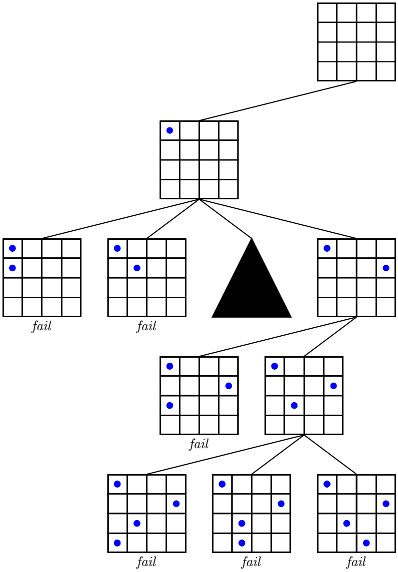
4-queens (backtracking)



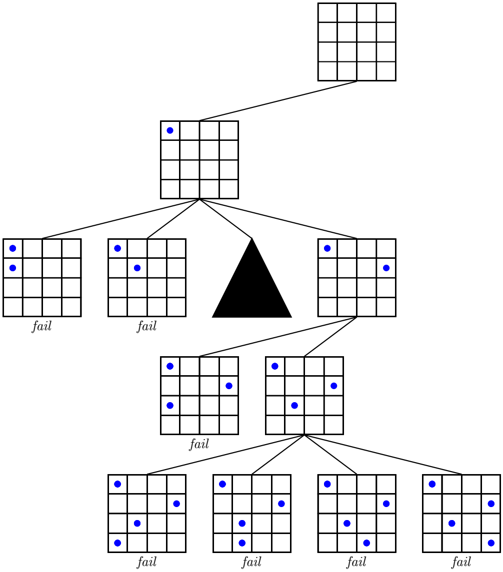
4-queens (backtracking)



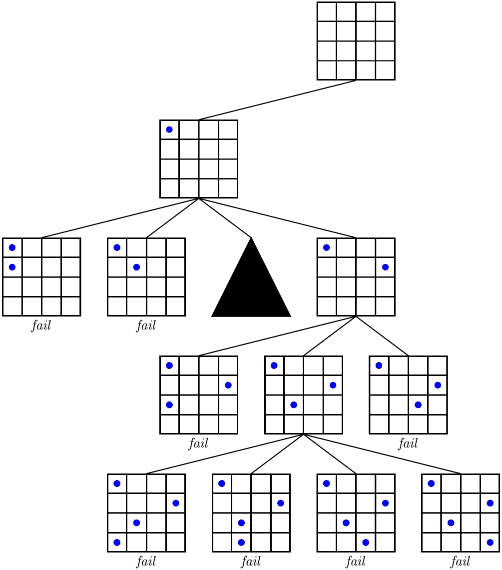
4-queens (backtracking)



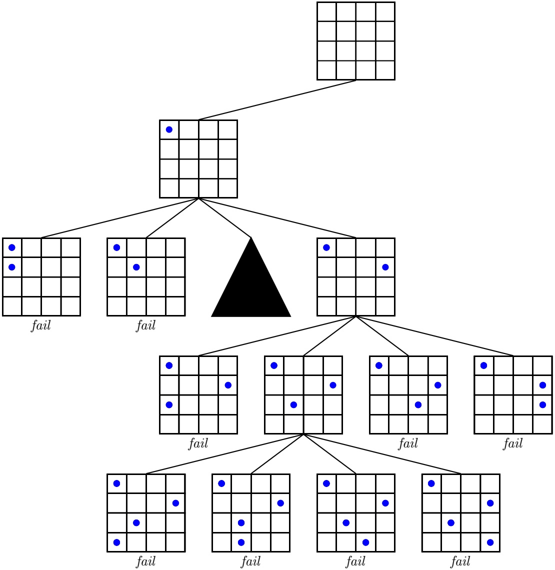
4-queens (backtracking)



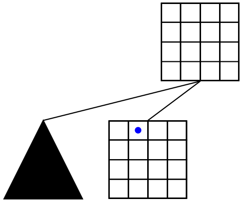
4-queens (backtracking)



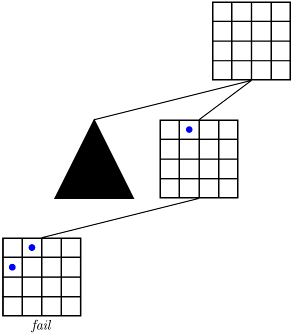
4-queens (backtracking)



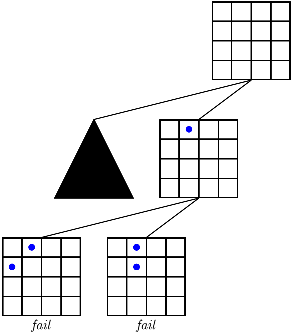
4-queens (backtracking)



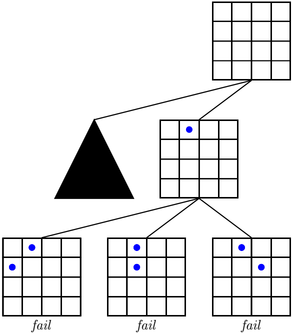
4-queens (backtracking)



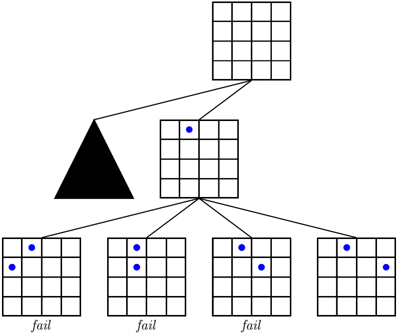
4-queens (backtracking)



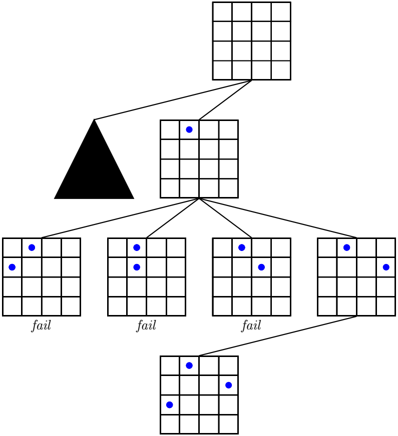
4-queens (backtracking)



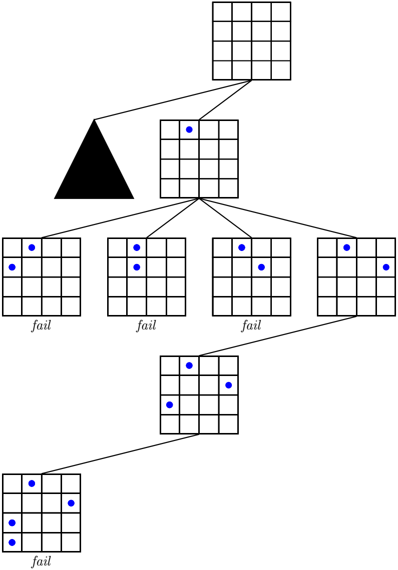
4-queens (backtracking)



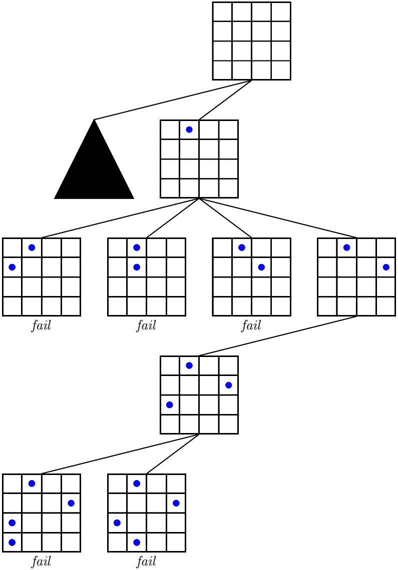
4-queens (backtracking)



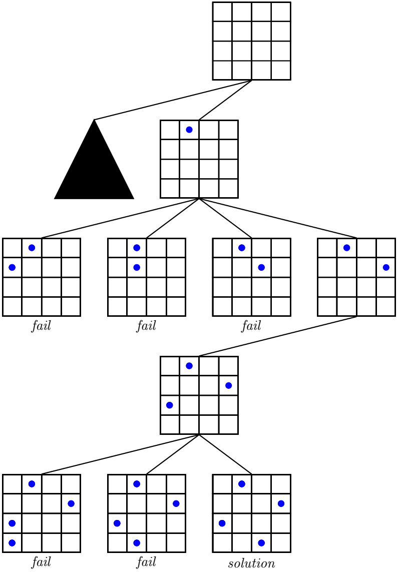
4-queens (backtracking)



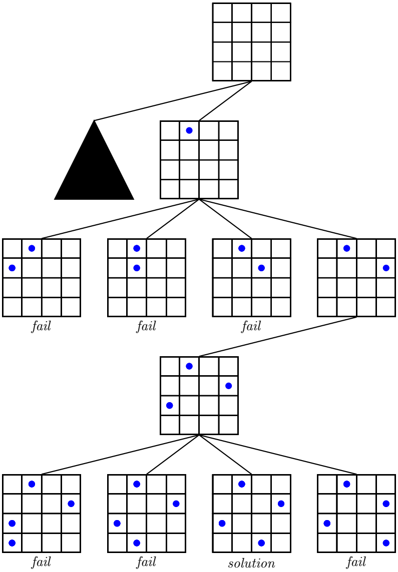
4-queens (backtracking)



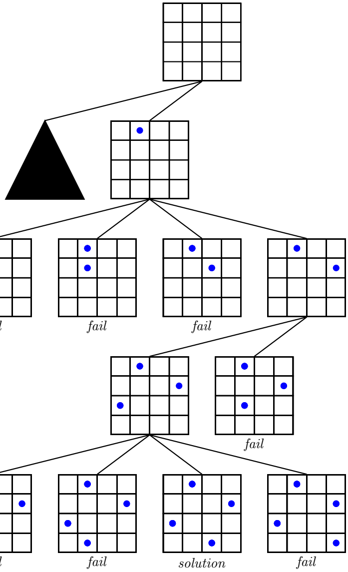
4-queens (backtracking)



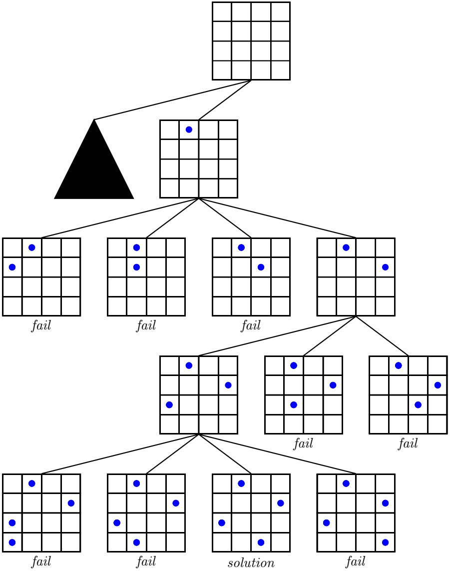
4-queens (backtracking)



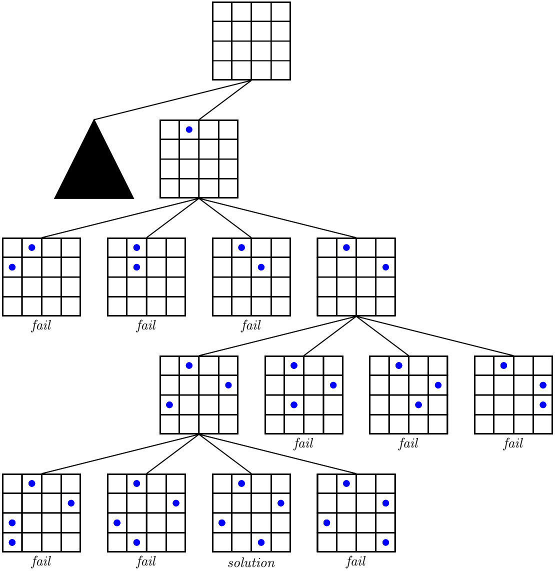
4-queens (backtracking)



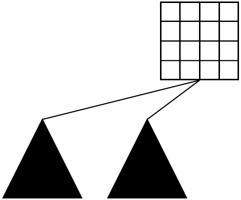
4-queens (backtracking)



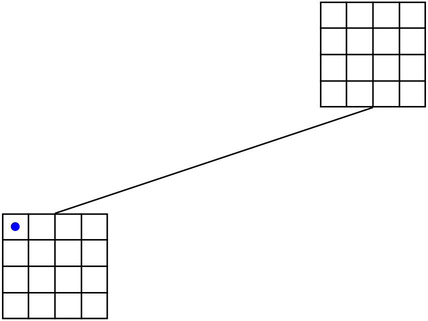
4-queens (backtracking)



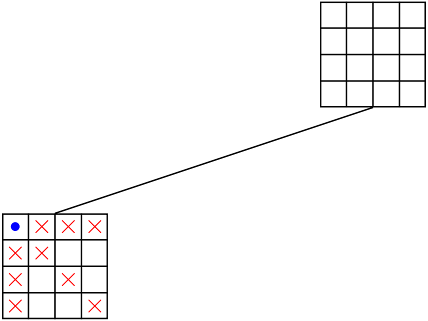
4-queens (backtracking)



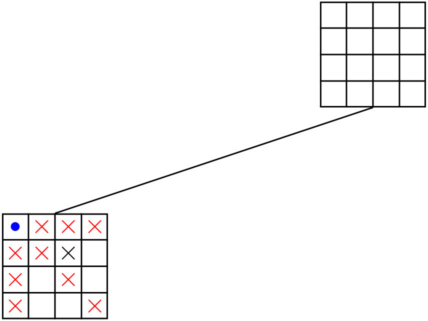
4-queens (MAC)



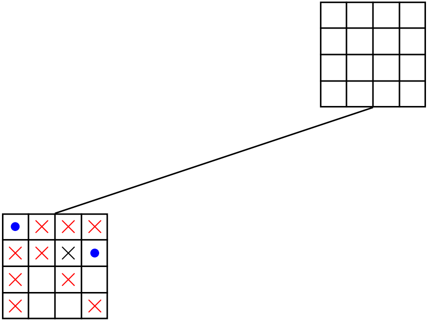
4-queens (MAC)



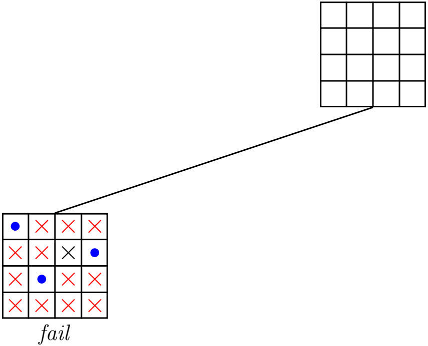
4-queens (MAC)



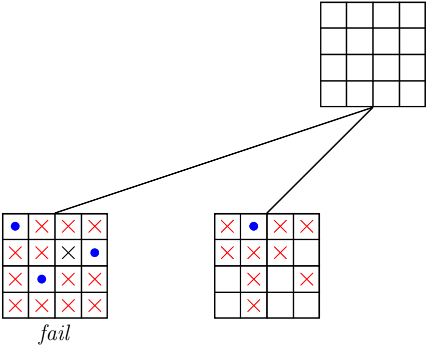
4-queens (MAC)



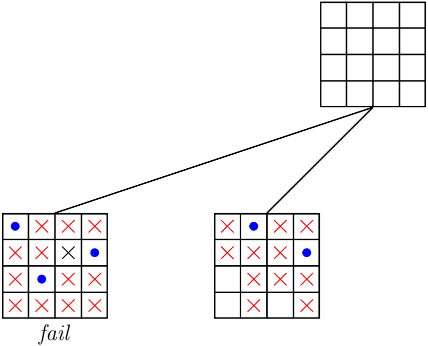
4-queens (MAC)



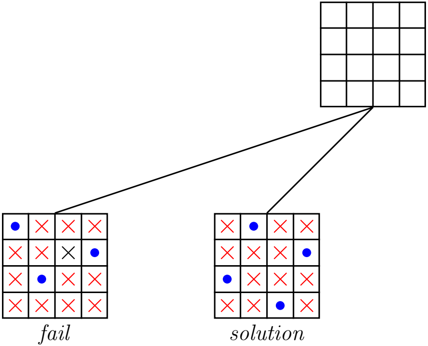
4-queens (MAC)



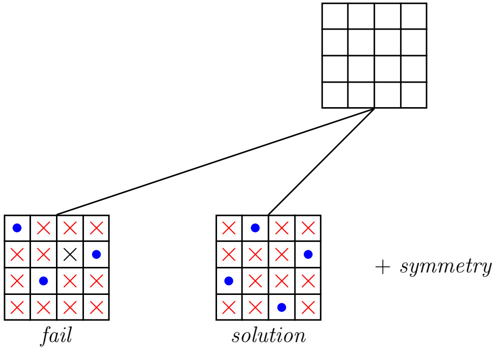
4-queens (MAC)



4-queens (MAC)



4-queens (MAC)



Result analysis

- The backtracking algorithm alone leads to 26 dead-end leaves.

Result analysis

- The backtracking algorithm alone leads to 26 dead-end leaves.
- Maintaining AC at each node leads to only 2 dead-end leaves.

Result analysis

- The backtracking algorithm alone leads to 26 dead-end leaves.
- Maintaining AC at each node leads to only 2 dead-end leaves.
- However, the complete search algorithm remains exponential in the problem size in the worst case.

Result analysis

- The backtracking algorithm alone leads to 26 dead-end leaves.
- Maintaining AC at each node leads to only 2 dead-end leaves.
- However, the complete search algorithm remains exponential in the problem size in the worst case.
- The goal is to balance the search and propagation efforts in order to reach a good practical complexity.

Generalized arc consistency (GAC)

Definition

Let c be a constraint on $X_c = \{x_{i_1}, \ldots, x_{i_k}\}$ and let $D_{i_1}, \ldots, D_{i_k}$ be their domains. The variable x_{i_1} is arc consistent relative to c if and only if

$$\forall a_{i_1} \in D_1 \; \exists a_{i_2} \in D_{i_2} \; \cdots \; \exists a_{i_k} \in D_{i_k} : (a_{i_1}, \ldots, a_{i_k}) \in R_c.$$

Generalized arc consistency (GAC)

Definition

Let c be a constraint on $X_c = \{x_{i_1}, \ldots, x_{i_k}\}$ and let $D_{i_1}, \ldots, D_{i_k}$ be their domains. The variable x_{i_1} is arc consistent relative to c if and only if

$$\forall a_{i_1} \in D_1 \; \exists a_{i_2} \in D_{i_2} \; \cdots \; \exists a_{i_k} \in D_{i_k} : (a_{i_1}, \ldots, a_{i_k}) \in R_c.$$

- GAC is an extension of AC for non binary constraints.

Generalized arc consistency (GAC)

Definition

Let c be a constraint on $X_c = \{x_{i_1}, \ldots, x_{i_k}\}$ and let $D_{i_1}, \ldots, D_{i_k}$ be their domains. The variable x_{i_1} is arc consistent relative to c if and only if

$$\forall a_{i_1} \in D_1 \; \exists a_{i_2} \in D_{i_2} \; \cdots \; \exists a_{i_k} \in D_{i_k} : (a_{i_1}, \ldots, a_{i_k}) \in R_c.$$

- GAC is an extension of AC for non binary constraints.
- The Revise procedure must examine the product of domains in the worst case, leading to a complexity of $O(d^k)$.

Domain consistency

- NC, AC and GAC are known as domain consistency.

Domain consistency

- NC, AC and GAC are known as domain consistency.
- Every value from every domain is examined.

Domain consistency

- NC, AC and GAC are known as domain consistency.
- Every value from every domain is examined.
- Every value must have a domain support.

Domain consistency

- NC, AC and GAC are known as domain consistency.
- Every value from every domain is examined.
- Every value must have a domain support.
- A domain support is a combination of values from the other domains ensuring constraint satisfaction.

Domain consistency

- NC, AC and GAC are known as domain consistency.
- Every value from every domain is examined.
- Every value must have a domain support.
- A domain support is a combination of values from the other domains ensuring constraint satisfaction.
- But enforcing domain consistency is too expensive in general.

Motivation

Ordering

We assume that the domains are totally ordered.

Motivation

Ordering

We assume that the domains are totally ordered.

Intervals

Domains are represented as intervals of integers.

$$\begin{aligned} D_x &= [\min(D_x), \max(D_x)] \\ &= \{\min(D_x), \min(D_x) + 1, \ldots, \max(D_x)\} \end{aligned}$$

Motivation

Ordering

We assume that the domains are totally ordered.

Intervals

Domains are represented as intervals of integers.

$$\begin{aligned} D_x &= [\min(D_x), \max(D_x)] \\ &= \{\min(D_x), \min(D_x)+1, \ldots, \max(D_x)\} \end{aligned}$$

Remark

It is not possible to dig holes in domains $\Longrightarrow$

Motivation

Ordering

We assume that the domains are totally ordered.

Intervals

Domains are represented as intervals of integers.

$$\begin{aligned} D_x &= [\min(D_x), \max(D_x)] \\ &= \{\min(D_x), \min(D_x)+1, \ldots, \max(D_x)\} \end{aligned}$$

Remark

It is not possible to dig holes in domains $\Longrightarrow$ only domain bounds must be examined.

Bounds consistency

Definition

Let c be a constraint on $X_c = \{x_{i_1}, \ldots, x_{i_k}\}$ and let $D_{i_1}, \ldots, D_{i_k}$ be their domains. The variable x_{i_1} is bounds consistent relative to c if and only if

$$\forall a_{i_1} \in \{\min(D_1), \max(D_1)\}$$

$$\exists a_{i_2} \in D_{i_2} \;\; \cdots \;\; \exists a_{i_k} \in D_{i_k} : (a_{i_1}, \ldots, a_{i_k}) \in R_c.$$

Bounds consistency

Definition

Let c be a constraint on $X_c = \{x_{i_1}, \ldots, x_{i_k}\}$ and let $D_{i_1}, \ldots, D_{i_k}$ be their domains. The variable x_{i_1} is bounds consistent relative to c if and only if

$$\forall a_{i_1} \in \{\min(D_1), \max(D_1)\}$$

$$\exists a_{i_2} \in D_{i_2} \ \cdots \ \exists a_{i_k} \in D_{i_k} : (a_{i_1}, \ldots, a_{i_k}) \in R_c.$$

Given $D_1 = [-1, 1]$ *prove that* x_1 *is bounds consistent relative to* $c : x^2 - 1 = 0$*. Is it node consistent?*

Checking bounds consistency

- Given a domain bound, checking the existence of a support runs in $O(d^{k-1})$ in the worst case...

Checking bounds consistency

- Given a domain bound, checking the existence of a support runs in $O(d^{k-1})$ in the worst case...
- But some specific algorithms are much more efficient.
  - arithmetic constraints
  - all different constraint

Primitive (addition) constraint

Theorem

Let the constraint $c : x_3 = x_1 + x_2$ with domains D_1, D_2, D_3.
Suppose that the following conditions are verified:

$$
\begin{array}{llcll}
i. & \min(D_3) & \geq & \min(D_1) & + & \min(D_2) \\
ii. & \max(D_3) & \leq & \max(D_1) & + & \max(D_2) \\
iii. & \min(D_1) & \geq & \min(D_3) & - & \max(D_2) \\
iv. & \max(D_1) & \leq & \max(D_3) & - & \min(D_2) \\
v. & \min(D_2) & \geq & \min(D_3) & - & \max(D_1) \\
vi. & \max(D_2) & \leq & \max(D_3) & - & \min(D_1)
\end{array}
$$

Then,

Primitive (addition) constraint

Theorem

Let the constraint $c : x_3 = x_1 + x_2$ with domains D_1, D_2, D_3.
Suppose that the following conditions are verified:

$$\begin{array}{llcll}
i. & \min(D_3) & \geq & \min(D_1) & + & \min(D_2) \\
ii. & \max(D_3) & \leq & \max(D_1) & + & \max(D_2) \\
iii. & \min(D_1) & \geq & \min(D_3) & - & \max(D_2) \\
iv. & \max(D_1) & \leq & \max(D_3) & - & \min(D_2) \\
v. & \min(D_2) & \geq & \min(D_3) & - & \max(D_1) \\
vi. & \max(D_2) & \leq & \max(D_3) & - & \min(D_1)
\end{array}$$

Then,

1. every x_i is bounds consistent relative to c, and moreover

Primitive (addition) constraint

Theorem

Let the constraint $c : x_3 = x_1 + x_2$ with domains D_1, D_2, D_3.
Suppose that the following conditions are verified:

$$
\begin{array}{lllll}
i. & \min(D_3) & \geq & \min(D_1) & + & \min(D_2) \\
ii. & \max(D_3) & \leq & \max(D_1) & + & \max(D_2) \\
iii. & \min(D_1) & \geq & \min(D_3) & - & \max(D_2) \\
iv. & \max(D_1) & \leq & \max(D_3) & - & \min(D_2) \\
v. & \min(D_2) & \geq & \min(D_3) & - & \max(D_1) \\
vi. & \max(D_2) & \leq & \max(D_3) & - & \min(D_1)
\end{array}
$$

Then,

1. every x_i is bounds consistent relative to c, and moreover
2. every x_i is domain consistent relative to c.

Primitive constraint (cont)

Proof (domain consistency of x_3)

Let $a_3 \in D_3$ and suppose that $a_3 \neq a_1 + a_2$ for all $a_1 \in D_1$ and for all $a_2 \in D_2$.

Primitive constraint (cont)

Proof (domain consistency of x_3)

Let $a_3 \in D_3$ and suppose that $a_3 \neq a_1 + a_2$ for all $a_1 \in D_1$ and for all $a_2 \in D_2$.

- Since $a_1 \in D_1$ and $a_2 \in D_2$ it comes that

$$\min(D_1) + \min(D_2) \leq a_1 + a_2 \leq \max(D_1) + \max(D_2).$$

Primitive constraint (cont)

Proof (domain consistency of x_3)

Let $a_3 \in D_3$ and suppose that $a_3 \neq a_1 + a_2$ for all $a_1 \in D_1$ and for all $a_2 \in D_2$.

- Since $a_1 \in D_1$ and $a_2 \in D_2$ it comes that

$$\min(D_1) + \min(D_2) \leq a_1 + a_2 \leq \max(D_1) + \max(D_2).$$

- Since $a_3 \neq a_1 + a_2$ and the addition is *interval-convex*, there are two cases.
  - Case 1: $a_3 < \min(D_1) + \min(D_2)$. From (i) it follows that $a_3 < \min(D_3)$, which is a contradiction.

Primitive constraint (cont)

Proof (domain consistency of x_3)

Let $a_3 \in D_3$ and suppose that $a_3 \neq a_1 + a_2$ for all $a_1 \in D_1$ and for all $a_2 \in D_2$.

- Since $a_1 \in D_1$ and $a_2 \in D_2$ it comes that

$$\min(D_1) + \min(D_2) \leq a_1 + a_2 \leq \max(D_1) + \max(D_2).$$

- Since $a_3 \neq a_1 + a_2$ and the addition is *interval-convex*, there are two cases.
  - Case 1: $a_3 < \min(D_1) + \min(D_2)$. From (i) it follows that $a_3 < \min(D_3)$, which is a contradiction.
  - Case 2: $a_3 > \max(D_1) + \max(D_2)$. From (ii) it follows that $a_3 > \max(D_3)$, which is a contradiction.

Primitive constraint (cont)

Proof (domain consistency of x_3)

Let $a_3 \in D_3$ and suppose that $a_3 \neq a_1 + a_2$ for all $a_1 \in D_1$ and for all $a_2 \in D_2$.

- Since $a_1 \in D_1$ and $a_2 \in D_2$ it comes that

$$\min(D_1) + \min(D_2) \leq a_1 + a_2 \leq \max(D_1) + \max(D_2).$$

- Since $a_3 \neq a_1 + a_2$ and the addition is *interval-convex*, there are two cases.
  - Case 1: $a_3 < \min(D_1) + \min(D_2)$. From (i) it follows that $a_3 < \min(D_3)$, which is a contradiction.
  - Case 2: $a_3 > \max(D_1) + \max(D_2)$. From (ii) it follows that $a_3 > \max(D_3)$, which is a contradiction.

Then x_3 is domain consistent relative to c.

Domain pruning

- If $\min(D_3) < \min(D_1) + \min(D_2)$ (condition i is false) then the left bound of D_3 can be reduced as

$$\min(D_3) \leftarrow \min(D_1) + \min(D_2)$$

Domain pruning

- If $\min(D_3) < \min(D_1) + \min(D_2)$ (condition i is false) then the left bound of D_3 can be reduced as

$$\min(D_3) \leftarrow \min(D_1) + \min(D_2)$$

- Hence it comes

$$\min(D_3) \leftarrow \max(\min(D_3), \min(D_1) + \min(D_2))$$

Domain pruning

- If $\min(D_3) < \min(D_1) + \min(D_2)$ (condition i is false) then the left bound of D_3 can be reduced as

$$\min(D_3) \leftarrow \min(D_1) + \min(D_2)$$

- Hence it comes

$$\min(D_3) \leftarrow \max(\min(D_3), \min(D_1) + \min(D_2))$$

- In a similar way, the right bound of D_3 can be reduced as

$$\max(D_3) \leftarrow \min(\max(D_3), \max(D_1) + \max(D_2))$$

Interval computations

Addition

$$[a, b] + [c, d] = [a + c, b + d]$$

Interval computations

Addition

$$[a, b] + [c, d] = [a + c, b + d]$$

Intersection

$$[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$$

Interval computations

Addition

$$[a, b] + [c, d] = [a + c, b + d]$$

Intersection

$$[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$$

Domain pruning

Let the constraint $c : x_3 = x_1 + x_2$ with domains D_1, D_2, D_3. If D_3 is assigned to

$$D_3 \leftarrow D_3 \cap (D_1 + D_2)$$

then x_3 is maximally (domain, bounds) consistent relative to c.

Interval arithmetic

Operations

$$
\begin{array}{lcccl}
[a,b] & + & [c,d] & = & [a+c, b+d] \\
[a,b] & - & [c,d] & = & [a-d, b-c] \\
[a,b] & \times & [c,d] & = & [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)]
\end{array}
$$

Interval arithmetic

Operations

$$
\begin{array}{rcccl}
[a,b] & + & [c,d] & = & [a+c, b+d] \\
[a,b] & - & [c,d] & = & [a-d, b-c] \\
[a,b] & \times & [c,d] & = & [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)]
\end{array}
$$

Evaluate the following operations.

1. $[0,8] + [-2,5]$
2. $[0,8] - [-2,5]$
3. $[0,8] \times [-2,5]$

Interval convexity

Definition

A binary operation $\diamond$ is interval convex if for every intervals $[a, b]$ and $[c, d]$ the set

$$\{x \diamond y : x \in [c, d], y \in [a, b]\}$$

is an interval.

Interval convexity

Definition

A binary operation $\diamond$ is interval convex if for every intervals $[a, b]$ and $[c, d]$ the set

$$\{x \diamond y : x \in [c, d], y \in [a, b]\}$$

is an interval.

- The operations $+, -, \times$ are interval convex.

Interval convexity

Definition

A binary operation $\diamond$ is interval convex if for every intervals $[a, b]$ and $[c, d]$ the set

$$\{x \diamond y : x \in [c, d] \,, y \in [a, b]\}$$

is an interval.

- The operations $+, -, \times$ are interval convex.
- The division (to be defined) is not interval convex.

$$\frac{[4, 8]}{[-2, 4]} = \{x : x \leq -2\} \ \cup \ \{x : x \geq 1\}$$

Inversion

Contraction by inversion

Since $x_3 = x_1 + x_2$ is equivalent to $x_1 = x_3 - x_2$ and $x_2 = x_3 - x_1$, it comes the following contraction rules.

$$\begin{array}{rcl} D_1 & \leftarrow & D_1 \cap (D_3 - D_2) \\ D_2 & \leftarrow & D_2 \cap (D_3 - D_1) \\ D_3 & \leftarrow & D_3 \cap (D_1 + D_2) \end{array}$$

Inversion

Contraction by inversion

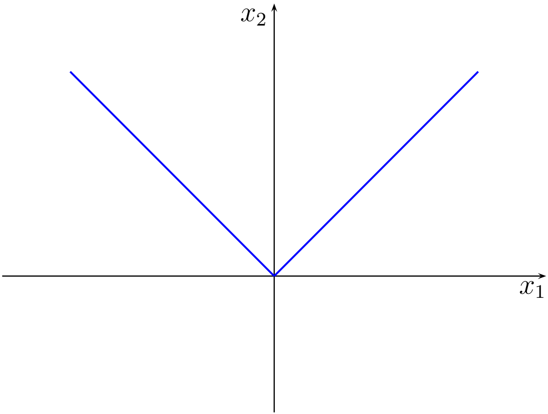
Since $x_3 = x_1 + x_2$ is equivalent to $x_1 = x_3 - x_2$ and $x_2 = x_3 - x_1$, it comes the following contraction rules.

$$\begin{array}{lcl} D_1 & \leftarrow & D_1 \cap (D_3 - D_2) \\ D_2 & \leftarrow & D_2 \cap (D_3 - D_1) \\ D_3 & \leftarrow & D_3 \cap (D_1 + D_2) \end{array}$$

$$\begin{array}{lcccccccc} D_1 & \leftarrow & [-3,2] & \cap & [0,4] & - & [-5,1] & = & [-1,2] \\ D_2 & \leftarrow & [-5,1] & \cap & [0,4] & - & [-3,2] & = & [-2,1] \\ D_3 & \leftarrow & [0,4] & \cap & [-3,2] & + & [-5,1] & = & [0,3] \end{array}$$

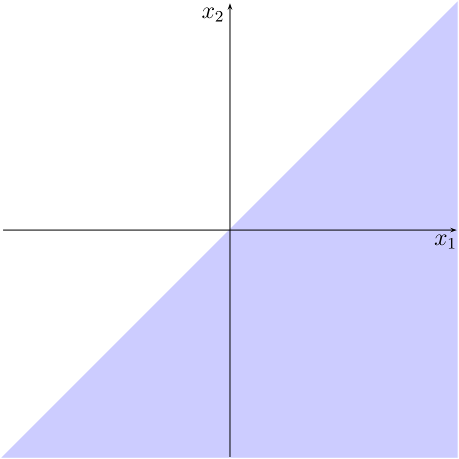
Inversion is not easy

Let the constraint $x_2 = |x_1|$. How to contract D_1, D_2?



Inequality constraint

Let the constraint $x_2 \leq x_1$. How to contract D_1, D_2?



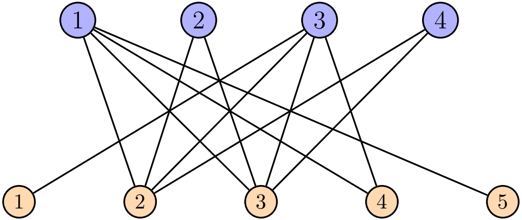
Decomposition of non primitive constraints

When a constraint involves more than one operation, it is possible to decompose it into an equivalent set of primitive constraints which can be processed by constraint propagation.

$$\boxed{\begin{array}{c} |x_1 x_2 - 1| \leq x_3 + 2 \\ \iff \\ \exists u_1 \exists u_2 \exists u_3 \exists u_4 \\ u_1 = x_1 x_2 \\ u_2 = u_1 - 1 \\ u_3 = \mathrm{abs}(u_2) \\ u_4 = x_3 + 2 \\ u_3 \leq u_4 \end{array}}$$

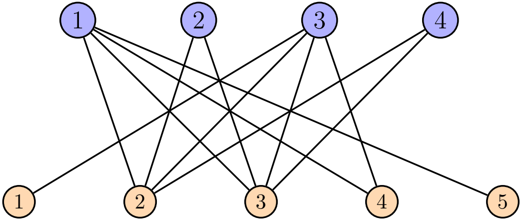
All different constraint

- Once again, we consider the task assignment problem (4 tasks and 5 machines).



All different constraint

- Once again, we consider the task assignment problem (4 tasks and 5 machines).



- Since $D_2 = D_4 = [2, 3]$ then machines 2 and 3 can be removed from task domains D_1 and D_3.

Hall theorem

Theorem

The constraint $\text{alldifferent}(x_1, \ldots, x_n)$ has a solution if and only if $|S| \leq |D_S|$ for all $S \subseteq \{x_1, \ldots, x_n\}$ where

$$D_S = \bigcup_{x_i \in S} D_i.$$

Hall theorem

Theorem

The constraint $\mathrm{alldifferent}(x_1, \ldots, x_n)$ has a solution if and only if $|S| \leq |D_S|$ for all $S \subseteq \{x_1, \ldots, x_n\}$ where

$$D_S = \bigcup_{x_i \in S} D_i.$$

- Given $D_1 = \{1, 2\}$, $D_2 = \{2, 3, 4\}$, $D_3 = \{2, 3\}$, $D_4 = \{1, 3\}$, $\mathrm{alldifferent}(x_1, x_2, x_3, x_4)$ has a solution.

Hall theorem

Theorem

The constraint $\mathrm{alldifferent}(x_1, \ldots, x_n)$ has a solution if and only if $|S| \leq |D_S|$ for all $S \subseteq \{x_1, \ldots, x_n\}$ where

$$D_S = \bigcup_{x_i \in S} D_i.$$

- Given $D_1 = \{1, 2\}$, $D_2 = \{2, 3, 4\}$, $D_3 = \{2, 3\}$, $D_4 = \{1, 3\}$, $\mathrm{alldifferent}(x_1, x_2, x_3, x_4)$ has a solution.
- Given $D_1 = \{1, 2, 4\}$, $D_2 = \{2, 3\}$, $D_3 = \{2, 3\}$, $D_4 = \{2, 3\}$, $\mathrm{alldifferent}(x_1, x_2, x_3, x_4)$ has no solution.

Hall interval

Definition

Let the variables $x_1, x_2, \ldots, x_n$ with domains $D_1, D_2, \ldots, D_n$.

Let I be an interval, and define $S_I = \{x_k : D_k \subseteq I\}$.

I is a Hall interval if $|I| = |S_I|$.

Hall interval

Definition

Let the variables $x_1, x_2, \ldots, x_n$ with domains $D_1, D_2, \ldots, D_n$.

Let I be an interval, and define $S_I = \{x_k : D_k \subseteq I\}$.

I is a Hall interval if $|I| = |S_I|$.

- If $|I| = |S_I|$ then all the values in I must be assigned to the variables in S_I.

Hall interval

Definition

Let the variables $x_1, x_2, \ldots, x_n$ with domains $D_1, D_2, \ldots, D_n$.

Let I be an interval, and define $S_I = \{x_k : D_k \subseteq I\}$.

I is a Hall interval if $|I| = |S_I|$.

- If $|I| = |S_I|$ then all the values in I must be assigned to the variables in S_I.
- These values can be removed from domains of the other variables.

Bounds consistency

Theorem

The constraint $\mathrm{alldifferent}(x_1, \ldots, x_n)$ is bounds consistent if and only if

1. $|D_k| \geq 1$ for $k = 1, \ldots, n$,

Bounds consistency

Theorem

The constraint $\text{alldifferent}(x_1, \ldots, x_n)$ is bounds consistent if and only if

1. $|D_k| \geq 1$ for $k = 1, \ldots, n$,
2. $|S_I| \leq |I|$ for each interval I,

Bounds consistency

Theorem

The constraint $\text{alldifferent}(x_1, \ldots, x_n)$ is bounds consistent if and only if

1. $|D_k| \geq 1$ for $k = 1, \ldots, n$,
2. $|S_I| \leq |I|$ for each interval I,
3. for each Hall interval I, for all $x_k \notin S_I$

$$\min(D_k) \notin I \quad \text{and} \quad \max(D_k) \notin I.$$

Revise procedure

```
ReviseAllDifferent(c(x_1,...,x_n): constraint,
                   D_1,...,D_n: domain)
begin
    let D = D_1 ∪ ··· ∪ D_n
    for each interval I ⊆ D do
        calculate S_I
        if |I| < |S_I| then
            fail
        else if |I| = |S_I| then
            for each variable x_k ∉ S_I
              update the bounds of D_k
            endfor
        endif
    endfor
end
```

Revise procedure (cont)

Complexity

$O(n|D|^2)$ updates of domains, since there are $O(|D|^2)$ intervals and $O(n)$ variables are considered in the internal loop.

Revise procedure (cont)

Complexity

$O(n|D|^2)$ updates of domains, since there are $O(|D|^2)$ intervals and $O(n)$ variables are considered in the internal loop.

Improvement

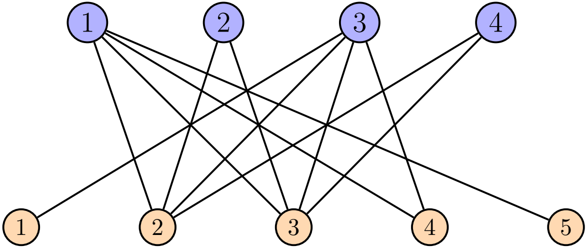
It is not necessary to consider all Hall intervals.

If $[l_1, u]$ and $[l_2, u]$ are both Hall intervals with $l_1 \leq l_2$ then every value that is removed by considering $[l_2, u]$ would be removed by considering $[l_1, u]$ $\implies$ $[l_2, u]$ is useless.

It is sufficient to consider the Hall interval having the smallest lower bound for a given upper bound. . . algorithm in $O(n \log n)$.

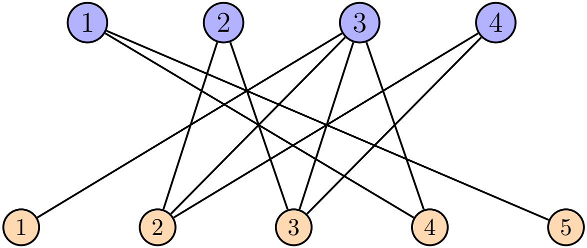
Example

Consider the task assignment problem.



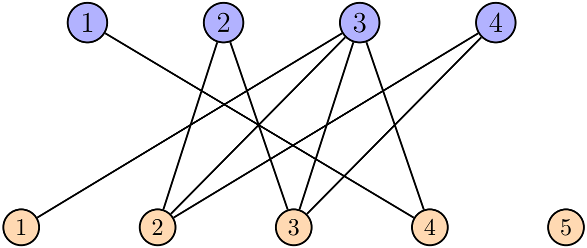
Example

D_1 is reduced since $[2, 3]$ is a Hall interval for $\{x_2, x_4\}$.



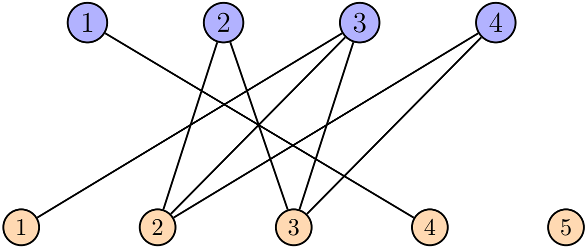
Example

Then x_1 is assigned to 4 (search).



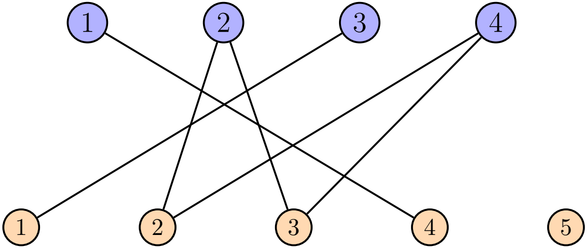
Example

D_3 is reduced since $[4,4]$ is a Hall interval for $\{x_1\}$.



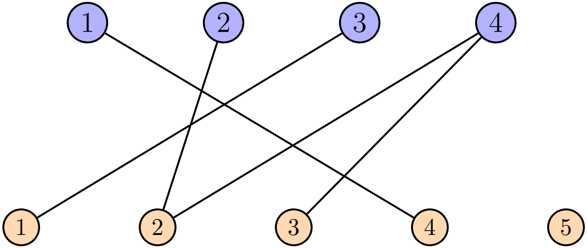
Example

D_3 is reduced since $[2, 3]$ is a Hall interval for $\{x_2, x_4\}$.



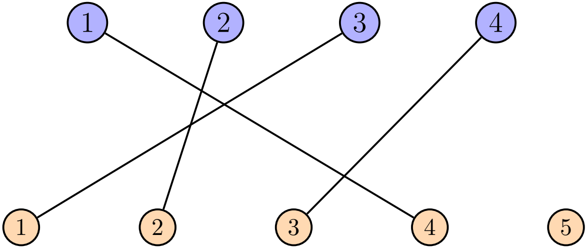
Example

Then x_2 is assigned to 2 (search).



Example

D_4 is reduced since $[2, 2]$ is a Hall interval for $\{x_2\}$.



This is a solution (maximum matching of size 4).

Then backtrack and find the other solutions (if required).

Bisection

New branching rule

- Let a search tree node with domains $(D_1, D_2, \ldots, D_n)$ and suppose that the variable x_1 has been selected.

Bisection

New branching rule

- Let a search tree node with domains $(D_1, D_2, \ldots, D_n)$ and suppose that the variable x_1 has been selected.
- Let the midpoint of D_1 be

$$c = \left\lfloor \frac{\min(D_1) + \max(D_1)}{2} \right\rfloor .$$

Bisection

New branching rule

- Let a search tree node with domains $(D_1, D_2, \ldots, D_n)$ and suppose that the variable x_1 has been selected.
- Let the midpoint of D_1 be

$$c = \left\lfloor \frac{\min(D_1) + \max(D_1)}{2} \right\rfloor .$$

- Then the following two sub-nodes are created:

$$([\min(D_1), c], D_2, \ldots, D_n) \quad ([c+1, \max(D_1)], D_2, \ldots, D_n)$$

Bisection

New branching rule

- Let a search tree node with domains $(D_1, D_2, \ldots, D_n)$ and suppose that the variable x_1 has been selected.
- Let the midpoint of D_1 be

$$c = \left\lfloor \frac{\min(D_1) + \max(D_1)}{2} \right\rfloor .$$

- Then the following two sub-nodes are created:

$$([\min(D_1), c], D_2, \ldots, D_n) \quad ([c+1, \max(D_1)], D_2, \ldots, D_n)$$

- Bisection can cope with large domains ($\neq$ unit split).

Bisection

New branching rule

- Let a search tree node with domains $(D_1, D_2, \ldots, D_n)$ and suppose that the variable x_1 has been selected.
- Let the midpoint of D_1 be

$$c = \left\lfloor \frac{\min(D_1) + \max(D_1)}{2} \right\rfloor .$$

- Then the following two sub-nodes are created:

$$([\min(D_1), c], D_2, \ldots, D_n) \quad ([c+1, \max(D_1)], D_2, \ldots, D_n)$$

- Bisection can cope with large domains ($\neq$ unit split).
- Bisection works with bounds consistency ($\neq$ forward checking).

Motivation

Numerical constraints

Constraints are built as usual from

- a countable set of variables,
- the set of real numbers $\mathbb{R}$,
- a set of operations $\{+, -, \times, \div, \mathrm{pow}, \log, \exp, \cos, \sin \ldots\}$,
- and a set of relations $\{=, \leq, \geq\}$.

Motivation

Numerical constraints

Constraints are built as usual from

- a countable set of variables,
- the set of real numbers $\mathbb{R}$,
- a set of operations $\{+, -, \times, \div, \mathrm{pow}, \log, \exp, \cos, \sin \dots\}$,
- and a set of relations $\{=, \leq, \geq\}$.

Intervals

Domains are represented as intervals of real numbers bounded by floating-point numbers (intervals representable in a machine).

Problems

Finite representation

The real number $a = 0.1$ is not representable.

$$a \to 0.099999999\ldots \quad \text{or} \quad a \to 0.1000000\ldots01.$$

Problems

Finite representation

The real number $a = 0.1$ is not representable.

$$a \rightarrow 0.099999999\ldots \quad \text{or} \quad a \rightarrow 0.1000000\ldots01.$$

Rounding errors

The result of an operation may not be a float.

$$\frac{1.0}{10.0} = 0.1$$

Problems

Finite representation

The real number $a = 0.1$ is not representable.

$$a \to 0.099999999\ldots \quad \text{or} \quad a \to 0.1000000\ldots01.$$

Rounding errors

The result of an operation may not be a float.

$$\frac{1.0}{10.0} = 0.1$$

Nonlinear constraint solving

$$x_n\left(x_k + \sum_{i=1}^{n-k-1} x_i x_{i+k}\right) = k,\ \ 1 \leq k < n$$

Floating point intervals

- Floating point numbers (set $\mathbb{F}$)
  - finite set of rational numbers
  - two zeros $0^-, 0^+$
  - two infinities $-\infty, +\infty$
  - NaN (not a number)

Floating point intervals

- Floating point numbers (set $\mathbb{F}$)
  - finite set of rational numbers
  - two zeros $0^-, 0^+$
  - two infinities $-\infty, +\infty$
  - NaN (not a number)
  - rounding downward (mapping $\mathbb{R} \to \mathbb{F}$)

$$r \mapsto \downarrow r \downarrow = \max\{a \in \mathbb{F} : a \leq r\}$$

Floating point intervals

- Floating point numbers (set $\mathbb{F}$)
  - finite set of rational numbers
  - two zeros $0^-, 0^+$
  - two infinities $-\infty, +\infty$
  - NaN (not a number)
  - rounding downward (mapping $\mathbb{R} \to \mathbb{F}$)

$$r \ \mapsto \downarrow r \downarrow = \ \max\{a \in \mathbb{F} : a \leq r\}$$

  - rounding upward (mapping $\mathbb{R} \to \mathbb{F}$)

$$r \ \mapsto \uparrow r \uparrow = \ \min\{a \in \mathbb{F} : a \geq r\}$$

Floating point intervals

- Floating point numbers (set $\mathbb{F}$)
  - finite set of rational numbers
  - two zeros $0^-, 0^+$
  - two infinities $-\infty, +\infty$
  - NaN (not a number)
  - rounding downward (mapping $\mathbb{R} \to \mathbb{F}$)

$$r \mapsto \downarrow r \downarrow = \max\{a \in \mathbb{F} : a \leq r\}$$

  - rounding upward (mapping $\mathbb{R} \to \mathbb{F}$)

$$r \mapsto \uparrow r \uparrow = \min\{a \in \mathbb{F} : a \geq r\}$$

- Floating point intervals (set $\mathbb{IF}$)

$$[a, b] = \{r \in \mathbb{R} : a \in \mathbb{F},\ b \in \mathbb{F},\ a \leq r \leq b\}$$

Interval arithmetic

Operations

$$
\begin{aligned}
[a,b] + [c,d] &= [\downarrow a + c \downarrow, \uparrow b + d \uparrow] \\
[a,b] - [c,d] &= [\downarrow a - d \downarrow, \uparrow b - c \uparrow] \\
[a,b] \times [c,d] &= [\downarrow \min(ac, ad, bc, bd) \downarrow, \uparrow \max(ac, ad, bc, bd) \uparrow]
\end{aligned}
$$

Interval arithmetic

Operations

$$\begin{array}{rcl}
[a, b] + [c, d] & = & [\downarrow a + c \downarrow, \uparrow b + d \uparrow] \\
[a, b] - [c, d] & = & [\downarrow a - d \downarrow, \uparrow b - c \uparrow] \\
[a, b] \times [c, d] & = & [\downarrow \min(ac, ad, bc, bd) \downarrow, \uparrow \max(ac, ad, bc, bd) \uparrow]
\end{array}$$

- The resulting bounds are rounded towards the infinities.

Interval arithmetic

Operations

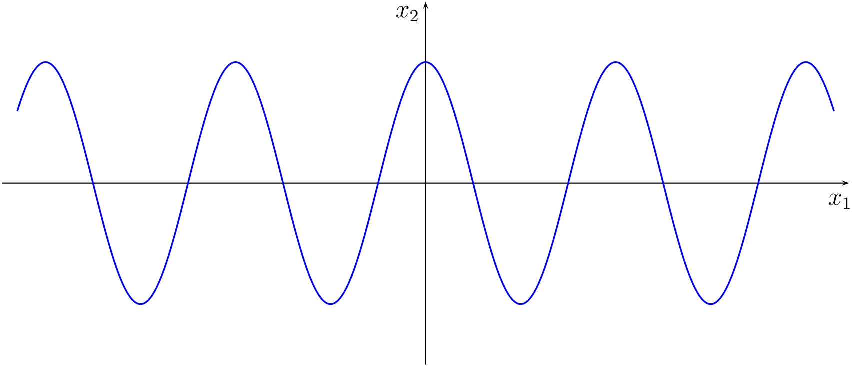
$$
\begin{aligned}
[a,b]+[c,d] &= [\downarrow a+c\downarrow, \uparrow b+d\uparrow] \\
[a,b]-[c,d] &= [\downarrow a-d\downarrow, \uparrow b-c\uparrow] \\
[a,b]\times[c,d] &= [\downarrow \min(ac,ad,bc,bd)\downarrow, \uparrow \max(ac,ad,bc,bd)\uparrow]
\end{aligned}
$$

- The resulting bounds are rounded towards the infinities.
- It implies the enclosure property.

$$\{x \diamond y : x \in [a,b]\,, y \in [c,d]\} \subseteq [a,b] \diamond [c,d]$$

Inversion may be hard

Example: $x_2 = \cos(x_1)$.



Interval methods

An efficient solving procedure may require several additionnal techniques.

- Fixed-point theorems

Interval methods

An efficient solving procedure may require several additionnal techniques.

- Fixed-point theorems
- Linearization (affine forms, centered forms)

Interval methods

An efficient solving procedure may require several additionnal techniques.

- Fixed-point theorems
- Linearization (affine forms, centered forms)
- Linear interval methods (preconditionned interval Gauss-Seidel)

Interval methods

An efficient solving procedure may require several additionnal techniques.

- Fixed-point theorems
- Linearization (affine forms, centered forms)
- Linear interval methods (preconditionned interval Gauss-Seidel)
- Strong consistency techniques (CID, 3B)

Interval methods

An efficient solving procedure may require several additionnal techniques.

- Fixed-point theorems
- Linearization (affine forms, centered forms)
- Linear interval methods (preconditionned interval Gauss-Seidel)
- Strong consistency techniques (CID, 3B)

. . . Global optimization course in M2!

Stopping criterion for search

Interval width

The width of an interval $[a, b]$ is the real number $\uparrow b - a \uparrow$.

Stopping criterion for search

Interval width

The width of an interval $[a, b]$ is the real number $\uparrow b - a \uparrow$.

Precision

Let the precision of computations $\epsilon > 0$ be a real number.

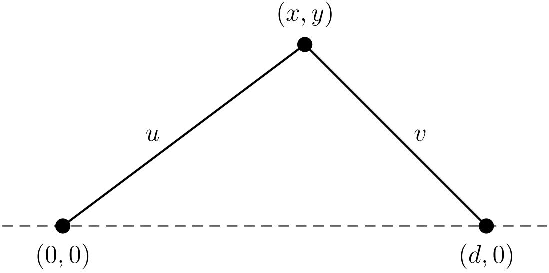
A node is declared to be final if either a domain is empty (failure) or the width of each domain is smaller then ϵ (success).

Kinematics closed loops

Non trivial kinematics closed loops (parallel robots) are applied in surgery, industry, flight simulation, etc.

A toy loop

Consider a simple kinematics closed loop.



Distance model

- There are 4 variables and 2 equations.

$$\begin{array}{ccccc} x^2 & + & y^2 & = & u^2 \\ (x-d)^2 & + & y^2 & = & v^2 \end{array}$$

- Domains of u and v are given.

$$u \in [2, 5.5],\ \ v \in [1.5, 7]$$

- This model is underconstrained (infinitely many solutions).

Inverse kinematics problem

Inverse kinematics

Fix the position x, y and find the lengths u, v.

Solving this problem is easy since there is a solved form.

$$\begin{aligned} u &= \sqrt{x^2 + y^2} \\ v &= \sqrt{(x-d)^2 + y^2} \end{aligned}$$

Result given $d = 8$, $x = 3.5$, $y = 2.5$:

$$u \in [4.3011626335213, 4.3011626335214]$$
$$v \in [5.1478150704935, 5.1478150704935]$$

Forward kinematics problem

Forward kinematics

Fix the lengths u, v and find the position x, y.

Solving this problem is hard in general since there is no solved form.

$$\begin{array}{cccccc} x^2 & + & y^2 & = & u^2 \\ (x-d)^2 & + & y^2 & = & v^2 \end{array}$$

Branch-and-prune

Solve the forward kinematics problem given $d = 8$, $u = 3$, $v = 5.25$.

$$D_x = [-10^8, 10^8], D_y = [-10^8, 10^8]$$

Branch-and-prune

Solve the forward kinematics problem given $d = 8$, $u = 3$, $v = 5.25$.

$$D_x = [-10^8, 10^8], D_y = [-10^8, 10^8]$$

$$\Big\downarrow \textit{propagation}$$

$$D_x = [2.750, 2.889], D_y = [-1.199, 1.199]$$

Branch-and-prune

Solve the forward kinematics problem given $d = 8$, $u = 3$, $v = 5.25$.

$$D_x = [-10^8, 10^8], D_y = [-10^8, 10^8]$$

propagation

$$D_x = [2.750, 2.889], D_y = [-1.199, 1.199]$$

bisection

$$D_x = [2.750, 2.889], D_y = [-1.199, 0] \qquad D_x = [2.750, 2.889], D_y = [0, 1.199]$$

Branch-and-prune

Solve the forward kinematics problem given $d = 8$, $u = 3$, $v = 5.25$.

$$D_x = [-10^8, 10^8], D_y = [-10^8, 10^8]$$

propagation

$$D_x = [2.750, 2.889], D_y = [-1.199, 1.199]$$

bisection

$$D_x = [2.750, 2.889], D_y = [-1.199, 0] \qquad D_x = [2.750, 2.889], D_y = [0, 1.199]$$

propagation

$$D_x = [2.839, 2.840], D_y = [-0.968, -0.967]$$

solution

Branch-and-prune

Solve the forward kinematics problem given $d = 8$, $u = 3$, $v = 5.25$.

$$D_x = [-10^8, 10^8], D_y = [-10^8, 10^8]$$

propagation

$$D_x = [2.750, 2.889], D_y = [-1.199, 1.199]$$

bisection

$$D_x = [2.750, 2.889], D_y = [-1.199, 0]$$

propagation

$$D_x = [2.839, 2.840], D_y = [-0.968, -0.967]$$

solution

$$D_x = [2.750, 2.889], D_y = [0, 1.199]$$

propagation

$$D_x = [2.839, 2.840], D_y = [0.967, 0.968]$$

solution