

Cours Master 2 Contraintes

1

Basic definitions

Constraint network

Definition 1 (Constraint network) A constraint network (or network) is composed of :

- a set of variables $X = \{X_1, \dots, X_n\}$
- a domain on X , that is, a set $D = \{D(X_1), \dots, D(X_n)\}$, where $D(X_i) \subset \mathbb{Z}$ is the finite set of values that variable X_i can take (its domain), and
- a set of constraints $C = \{c_1, \dots, c_e\}$

11

Constraint

Definition 2 (Constraint) A constraint c is a Boolean function involving a sequence of variables $X(c) = (X_{i_1}, \dots, X_{i_q})$ called its scheme. The function is defined on \mathbb{Z}^q . A combination of values (or tuple) $\tau \in \mathbb{Z}^q$ satisfies c if $c(\tau) = 1$ (also noted $\tau \in c$). If $c(\tau) = 0$ (or $\tau \notin c$), τ violates c .

Backtracking (BT)

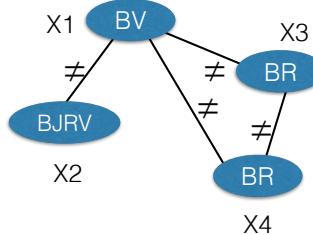
```

function BT( $N$  : problem ;  $I$  : instantiation) :Boolean
begin
    if  $|I| = n$  then return true termination
    choose a variable  $X_i$  not in  $I$  choose a variable  $X_i$  not in  $I$ 
    for each  $v_i \in D(X_i)$  do
        if  $I \cup \{X_i, v_i\}$  locally consistent then
            if BT( $N, I \cup \{X_i, v_i\}$ ) then return true
        return false recursive call
end

```

A call to BT(N, \emptyset) decides CSP

16



Local consistencies (constraint propagation)

19

Complexity of BT

- Space complexity: linear
- Time complexity:

$O(ed^n)$, where $n = |X|$, $e = |C|$, and $d = \max_{1..n}(D(X_i))$

- Repeatedly discovers the same inconsistencies
- Limited to very small problems

18

Arc consistency

Definition 4 (Arc consistency (AC)) Let $N = (X, D, C)$ a network and c a constraint in C .

- a tuple $\tau \in c \cap D^{X(c)}$ is called a support on c
- a value $v_i \in D(X_i)$ is viable iff $\forall c \in C, X_i \in X(c), \exists$ a support τ on c such that $\tau[X_i] = v_i$
- the domain D is arc consistent iff $\forall X_i \in X, \forall v_i \in D(X_i)$, v_i is viable
- the AC-closure of D is the domain D_{AC} which is obtained by iteratively removing non viable values from D until no more exists

D: $X_1 \ X_2 \ X_3$

1	1	1
2	2	2
3	3	3
7	7	7
8	8	8

$X_1 + X_2 = X_3$

1	1	2
1	2	3
2	1	3
1	7	8
7	1	8



D: $X_1 \ X_2 \ X_3$

1	1	1
2	2	2
3	3	3
7	7	7
8	8	8

24

AC3 [Mackworth1977]

```

function AC3(in X : set) : Boolean
begin
  1   Q ← {(Xi, c) | c ∈ C, Xi ∈ X(c)};
  2   while Q ≠ ∅ do
  3       select and delete (Xi, c) from Q;
  4       if revise(Xi, c) then
  5           if D(Xi) = ∅ then return false ;
  6           else Q ← Q ∪ {(Xj, c') | c' ∈ C ∧ {Xi, Xj} ⊆ X(c') ∧ j ≠ i};
  7   return true ;
end

```

26

Revise of AC3

```

function Revise3(in Xi : variable; cij : constraint) : Boolean
begin
  1   CHANGE ← false;
  2   foreach vi ∈ D(Xi) do
  3       vj ← first(D(Xj));
  4       while (vj ≠ nil) and ¬cij(vi, vj) do vj ← next(vj, D(Xj));
  5       if vj = nil then
  6           remove vi from D(Xi);
  7       CHANGE ← true;
  8   return CHANGE ;
end

```

28

Revise of AC2001

```

function Revise3(in Xi : variable; cij : constraint) : Boolean
begin
  1   CHANGE ← false;
  2   foreach vi ∈ D(Xi) do
  3       vj ← first(D(Xj));
  4       while (vj ≠ nil) and ¬cij(vi, vj) do vj ← next(vj, D(Xj));
  5       if vj = nil then
  6           remove vi from D(Xi);
  7       CHANGE ← true;
  8   return CHANGE ;
end

```

```

function Revise2001(in Xi : variable; cij : constraint) : Boolean
begin
  1   CHANGE ← false;
  2   foreach vi ∈ D(Xi) s.t. Last(Xi, vi, Xj) ∉ D(Xj) do
  3       vj ← next(Last(Xi, vi, Xj), D(Xj));
  4       while (vj ≠ nil) and c(vi, vj) ≠ cij do vj ← next(vj, D(Xj));
  5       if vj ≠ nil then Last(Xi, vi, Xj) ← vj;
  6       else
  7           remove vi from D(Xi);
  8           CHANGE ← true;
  9   return CHANGE ;
end

```

33

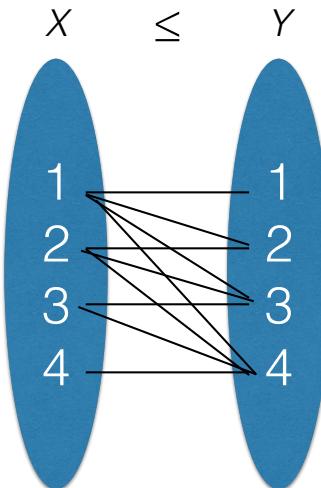
Complexity

- AC3
 - time $O(ed^3)$ on binary, space $O(e)$
- AC2001
 - time $O(ed^2)$ on binary ($O(ed^k)$ for k -ary), **optimal** space $O(ed)$
- STR
 - time $O(T)$, space $O(T)$ (T = taille de la table)

34

Removal events

- Each domain reduction is labeled with an ‘event’
 - **remValue**(X_i): a value is removed from $D(X_i)$
 - **incMin**(X_i): the smallest value is removed from $D(X_i)$
 - **decMax**(X_i): the greatest value is removed from $D(X_i)$
 - **instantiate**(X_i): $D(X_i)$ is reduced to a singleton

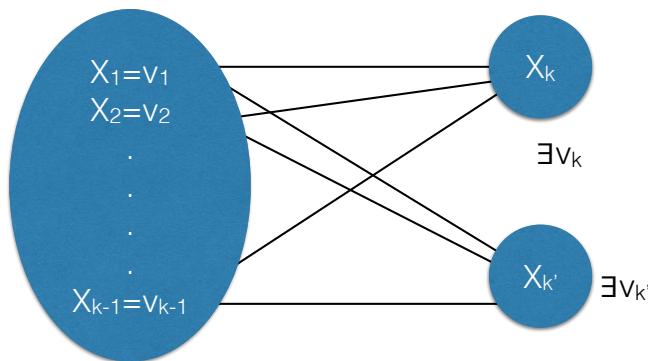


36

Stronger levels of consistency

39

k -consistency



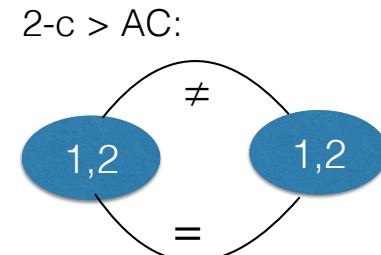
Definition 5 (k -consistency) A network N is k -consistent iff $\forall Y \subseteq X, |Y| = k - 1, \forall I$ locally consistent instantiation on $Y, \forall X_{i_k} \in X \setminus Y, \exists v_{i_k} \in D(X_{i_k})$ such that $I \cup (X_{i_k}, v_{i_k})$ is locally consistent.

40

2-consistency

	normalized network	non normalized
binary network	2-c = AC	2-c > AC
non binary	2-c < AC	2-c \neq AC

Normalized = not two constraints with the same scope



2-c < AC:
 $X+Y=Z$

1	1	1
2	2	2
3	3	3

42

Strong k -consistency

Definition 6 (Strong k -consistency) A network is strongly k -consistent iff it is j -consistent for any j , $1 \leq j \leq k$.

- time complexity is in $O(n^{k+1}d^{k+1})$ if space in $O(n^{k-1}d^{k-1})$
- time complexity is in $O(n^kd^k)$ if space in $O(n^kd^k)$

43

Global consistency

Definition 7 An instantiation is said globally consistent iff it can be extended to a solution. A network is said globally consistent iff every locally consistent instantiation is also globally consistent.

Observation:

A globally consistent network is obviously BT-free

Proposition 4 A network is globally consistent iff it is strongly n -consistent.

44

Speed-up Backtracking

(Complete) Search Algorithms

- look-back (reasoning on failure)
- look-ahead (constraint propagation)
- variable ordering
- value ordering

56

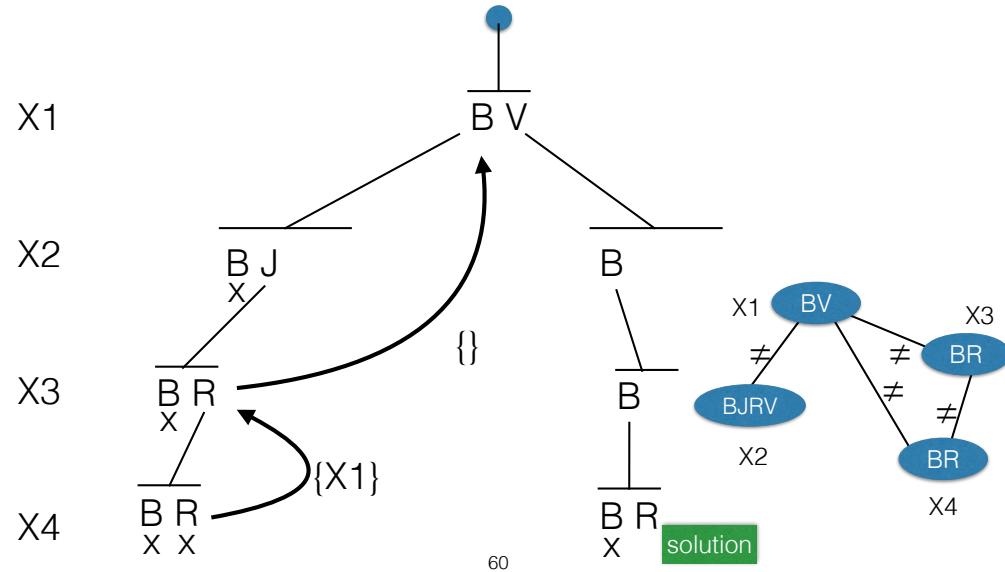
57

Backjumping

- Quand domaine vide : remonter à la variable la plus basse « impliquée dans l'échec »

59

CBJ on running example



60

Nogood learning

- A chaque domaine vide, on mémorise une raison de l'échec (nogood), par exemple : $[X_i \neq 2] \vee [X_j \neq 3]$
- Approche proposée dans les 80s sur les CSPs
 - lourd, peu efficace
- Récupéré par SAT en 1997 puis gros succès à partir de 2001 (1UIP et restart) —> solveurs CDCL
- Revenu dans les CSPs depuis 2007 (*lazy clause generation*). Nogoods plus expressifs : $[X_i \leq 6] \vee [X_j \neq 3]$.

61

Look-ahead (constraint propagation)

- Réfléchir d'abord où ne pas descendre, c'est à dire:
 - Enlever **des** branches qui conduisent à un échec
 - ❖ Instancier X_i
 - ❖ Pour tout $k > i$ supprimer de $D(X_k)$ **des** valeurs incompatibles avec $X_1.. X_i$
 - ❖ Si $D(X_k) = \emptyset$, arrêter cette branche

62

Forward checking (FC)

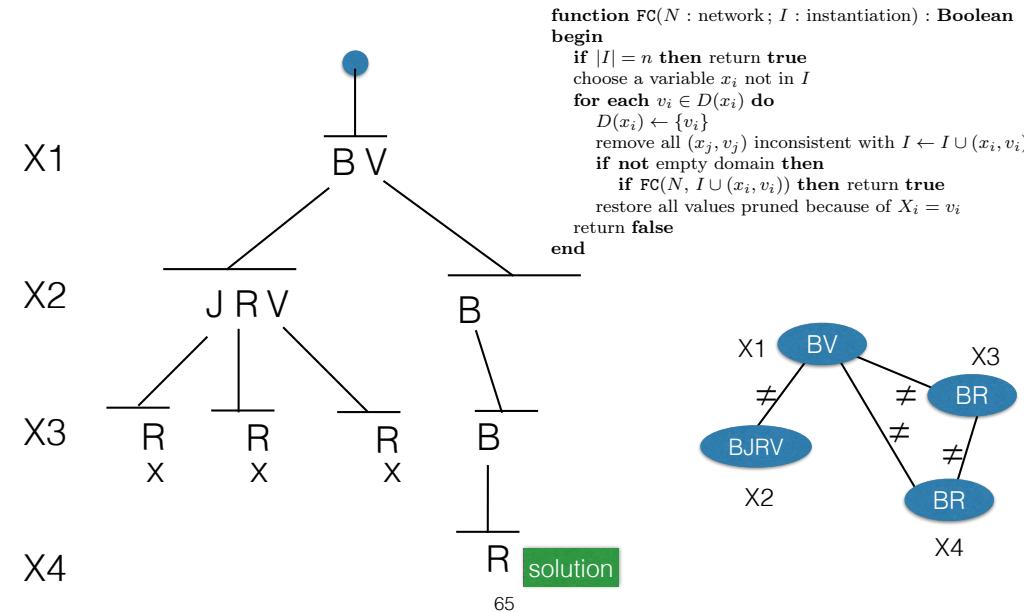
```

function FC( $N$  : network ;  $I$  : instantiation) : Boolean
begin
  if  $|I| = n$  then return true
  choose a variable  $x_i$  not in  $I$ 
  for each  $v_i \in D(x_i)$  do
     $D(x_i) \leftarrow \{v_i\}$ 
    remove all  $(x_j, v_j)$  inconsistent with  $I \leftarrow I \cup (x_i, v_i)$ 
    if not empty domain then
      if FC( $N, I \cup (x_i, v_i)$ ) then return true
      restore all values pruned because of  $X_i = v_i$ 
    return false
end

```

63

FC on running example



65

Maintaining Arc Consistency (MAC)

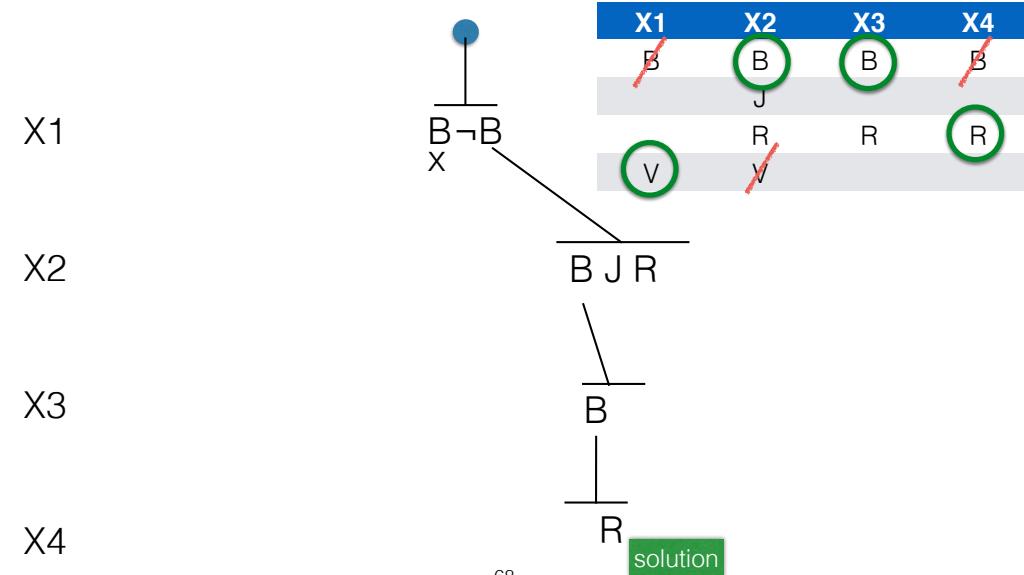
```

Function MAC( $N$ )
  AC( $N$ )
  if empty domain then return 0
  if  $N$  fully instantiated then return 1
  select a non singleton variable  $X_i$ 
  select a value  $v_i \in D(X_i)$ 
  return MAC( $N + \{X_i=v_i\}$ ) or MAC( $N + \{X_i \neq v_i\}$ )

```

67

MAC on running example



68

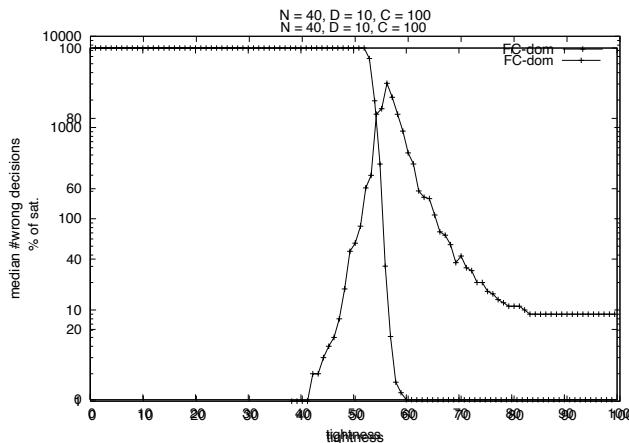
Random problems (binary)

- n : number of variables
- d : size of domains
- $p1$: proportion of constraints
 - number of constraints = $p1 \times n(n-1)/2$
- $p2$: tightness of constraints
 - number of forbidden tuples = $p2 \times d^2$

Parenthèse sur les expérimentations

71

Solving these instances



We observe a sudden increase of hardness at the phase transition

Variable ordering heuristics

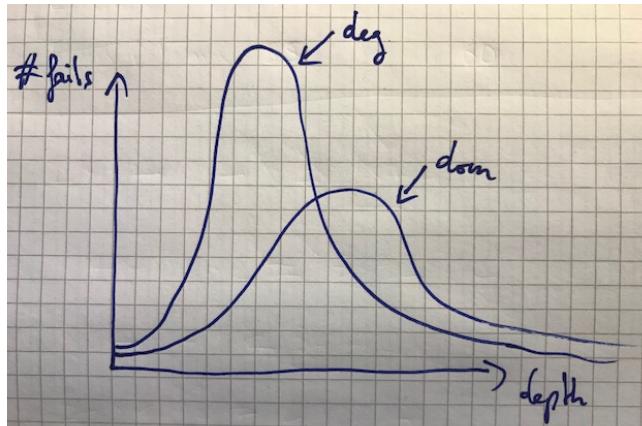
- We must instantiate **all** variables
 - start by the ‘most difficult’ variables
 - reduce depth of search tree (called *fail first principle*)

76

81

deg versus dom

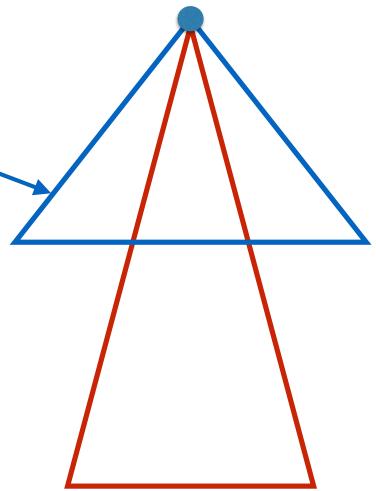
- deg reduces the **depth** of the search tree
- dom reduces the **width** of the search tree (branching factor)
- dom is good on dense networks, deg on sparse ones



87

Search tree shape

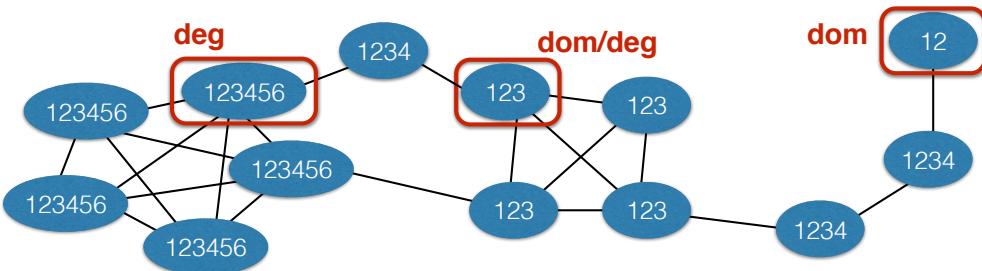
- difficult variables reduces the **depth** of the search tree
- small domains reduces the **width** of the search tree



88

Combining dom and deg

- **dom/deg**: select the variable X_i with smallest ratio: $\frac{|D(X_i)|}{|\Gamma(X_i)|}$



89

Value ordering heuristics

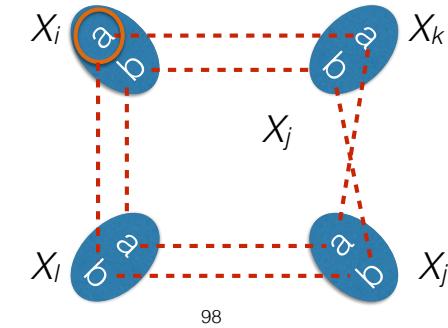
- We may not need to try all values of a variable to find a solution
→ start by the most promising
- Choosing a promising value is expensive (except if expert knowledge)
- On inconsistent problems value ordering is useless (Now, 95% of the time to solve hard problems is spent on inconsistent subtrees)
- → Use only if **expert knowledge available**

91

Singleton Arc Consistency (SAC)

Domain-based
consistencies stronger than AC

Definition 20 (Singleton arc consistency) A network $N = (X, D, C)$ is singleton arc consistent iff for all $X_i \in X$, for all $v_i \in D(X_i)$, the network $N + \{X_i = v_i\}$ is not arc inconsistent (i.e., $AC(N + \{X_i = v_i\}) \neq \emptyset$).



93

Global constraints

Definition 13 (Global constraint) A global constraint G is a class of constraints defined by a Boolean function f_G that takes as input an unbounded number of variables. That is, given n_0 , there exists an integer $n > n_0$ such that $\mathbb{Z}^n \cap f_G \neq \emptyset$.

Global constraints

Alldifferent(X_1, \dots, X_n):

all variables take different values ($X_i \neq X_j$ for all i, j)

Sum(X_1, \dots, X_n, N):

N is equal to the sum of the X_i 's ($\sum X_i = N$)

AtLeast-p-v(X_1, \dots, X_n):

at least p variables take value v ($|\{i \in 1..n \mid X_i = v\}| \geq p$)

Global constraints and propagation

- Global constraints compactly express properties of a problem
- Arc consistency allows strong propagation (because they preserve properties of the problem)
- **BUT:** Using a generic AC algorithm on $C(X_1 \dots X_n)$ is in $O(d^n)$
 - We need ad hoc propagators
- **BUT:** there are more than 400 global constraints in the catalog!
 - Decompose!

106

Simple Decompositions for Global Constraints

- A simple decomposition δ_G of a global constraint G is a function that given an instance $c(X)=(X,D_x,\{c\})$ of G returns a network $\delta_G(c)=(X,D_x,C)$ such that:
 - for all $c_i \in C$, c_i involves a **bounded** number of variables
 - $sol(c) = sol(\delta_G(c))$

107

Decompositions for Global Constraints

- A decomposition δ_G of a global constraint G is a function that given an instance $c(X)=(X,D_x,\{c\})$ of G returns a network $\delta_G(c)=(X,Y,D_x+D_y,C)$ such that:
 - for all $c_i \in C$, c_i involves a **bounded** number of variables
 - $sol(c) = sol(\delta_G(c))[X]$
 - $|Y| + |D_y|$ is polynomial in $|X| + |D_x|$

109

Do Decompositions Preserve Propagation?

- A decomposition δ_G of a global constraint G **preserves arc consistency** iff for any instance $c(X)=(X,D_x,\{c\})$ of G and any domain $D'_x \subseteq D_x$,
$$D'_{AC}(c) = D'_{AC}(\delta_G(c))[X]$$
- Constraint G is said **AC-decomposable**

111

Case 1: NP-hard constraints

Definition 17 Given a global constraint G , the problem checker_G is defined by :

- Instance : $(X(c), D, \{c\})$, where c is an instance of G
- Question : Is there a tuple $\tau \in D^{X(c)}$ such that $c(\tau)$?

- **Lemma** Given a constraint G , if checker_G is NP-complete on G , then arc consistency is NP-hard on G

Theorem 2 If a global constraint G is such that checker_G is NP-complete, then there doesn't exist any AC-decomposition of G .

- Examples: $\text{NValue}(X_1 \dots X_n, N)$, $\text{Sum}(X_1 \dots X_n, N)$, etc

115

Case 1: Apply weaker consistency
→ Bound consistency

- We relax the notion of support
- **Bound support:**
 - it is a tuple $t \in c(X)$ and for all $X_i \in X$, $\min(X_i) \leq t[X_i] \leq \max(X_i)$
 - We ensure bound support only for the min and max of each domain

116

Supports and Bound Supports

	support	bound support
all values	AC	Range Consistency
min and max	BC(D)	BC

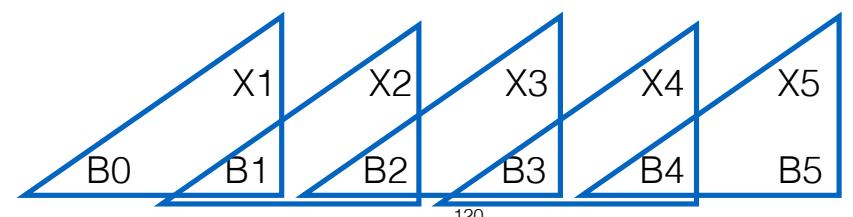
119

Case 2: AC-decomposable

Definition 18 The incidence graph of a network $N = (X, D, C)$ is the graph $G_N^I = (X \cup C, E)$ with $E = \{(x, c) \mid c \in C, x \in X(c)\}$. The network N is Berge-acyclic iff G_N^I is acyclic.

Theorem 3 If a network N is Berge-acyclic, all values in $AC(N)$ belong to a solution.

Corollary 2 If a global constraint has a decomposition that is Berge-acyclic, then this is an AC-decomposition.



120

The Picture

- Global constraints can be:
 - Type 1: We proved NP-hardness of arc consistency
 - Type 2: We found a decomposition preserving arc consistency
 - Is there a third type?YES!

122

Circuit Complexity

- If there exists a decomposition that preserves arc consistency on a global constraint c
 - $\Rightarrow \exists$ a poly-size CNF formula that decides existence of support on c
 - $\Rightarrow \exists$ a poly-size monotone Boolean circuit that decides existence of support on c

Theorem 4 *If there does not exist a poly-size monotone circuit for a Boolean function G then there does not exist any CNF decomposition computing AC on G and there does not exist any AC-decomposition for G .*

123

Case 3: polynomial and non AC-decomposable

- Some Boolean functions (such as “perfect matching”) cannot be represented as a monotone Boolean circuit [Razborov 1985]
- **Alldifferent** subsumes perfect matching
 - no decomposition preserves arc consistency on **Alldifferent**
 - no SAT formula for arc consistency on **Alldifferent**
- Other popular constraints: **global-cardinality**, **same**

124

Optimization

135

Dynamic CSP

- A set C_H of physical (hard) constraints and a set C_P of preferences (soft constraints)
- A sequence $N_0, N_1, \dots, N_i, \dots, N$, where $N_i = (X, D, C_i)$ with $C_i = C_{i-1} \pm \{c\}$, $c \in C_P$ and $C_0 = C_H$

137

maxCSP

- *Instance:* A constraint network $N = (X, D, C)$
- *Question:* Find an assignment on X that satisfies the **maximum number** of constraints
- Does not discriminate between hard and soft constraints

138

Soft CSPs / COPs

- Tackle optimization with standard CP solvers
- $c_j(X_1, \dots, X_k) \rightarrow \text{soft-}c_j(X_1, \dots, X_k, Y_j)$, where Y_j is the cost for c_j of the assignment on X_1, \dots, X_k

139

Cost Function Networks

- *Instance:* A cost function network $N = (X, D, F, k)$
- *Question:* Find an assignment I on X such that
$$\bigoplus_{f \in F} f(I[X(f)])$$
is **minimal** and strictly smaller than k .

$$a \oplus b = \min(a + b, k).$$

142

Project assignment

Dual models

- A set S of students and a set P of projects. Each student must specify the projects she would like to do
- Each project will be assigned to a single student
- Each project is proposed by a company
- A company cannot receive more than k students
- A set R of couples (i,j) of students who must be sent to companies in the same city

154

155

Dual model

- A variable X_i for each student in S **and** a variable Y_j for each project in P
- For each pair (i, j) in $S \times P$, a **channelling** constraint:

$$X_i = j \longleftrightarrow Y_j = i$$

- For each (i,j) in R, a constraint $c(X_i, X_j)$ ensures i and j will do projects in the same city
- For each set T of projects from a company, a constraint $\text{atleast}[|T|-k][0](Y[T])$

Polynomial-time constraint satisfaction problems

Clément Carbonnel

CNRS, LIRMM

06/11/2023

158

Part 1: language-based tractable classes

Fixed-language CSPs

A **constraint language** Γ is a finite set of Boolean functions over a finite domain D

$\text{CSP}(\Gamma)$ = CSP restricted to networks in which every constraint uses a function from Γ

- 3-SAT $\equiv \text{CSP}(\{\text{ternary Boolean clauses}\})$ over $D = \{0, 1\}$
- 3-COLORING $\equiv \text{CSP}(\{\neq\})$ over $D = \{1, 2, 3\}$
- etc.

What is the complexity of the following languages, over the domain $\{0, 1\}$?

- $\Gamma_0 = \{c_{\vee 3}\}$, where $c_{\vee 3}(x, y, z) = x \vee y \vee z$ P
- $\Gamma_1 = \{c_{\oplus}\}$, where $c_{\oplus}(x, y) = \{(0, 1), (1, 0)\}$ P
- $\Gamma_2 = \{c_{\vee 3}, c_{\oplus}\}$ NP-complete
- $\Gamma_3 = \{c_{1\text{-in-}3}\}$, where $c_{1\text{-in-}3}(x, y, z) = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$ NP-complete

pp-definitions

Definition

A **primitive-positive formula** (pp-formula) over a language Γ is a formula that only uses conjunction, existential quantification, equality and functions in Γ .

Definition

A function c is **primitive-positive definable** (pp-definable) over a language Γ if there exists a pp-formula ϕ over Γ such that

$$(d_1, \dots, d_r) \in c \iff \phi(d_1, \dots, d_r) \text{ is true}$$

Expressivity

Theorem (folklore)

If every function $c \in \Gamma_1$ is pp-definable over Γ_2 , then there is a (logspace) polynomial-time reduction from $\text{CSP}(\Gamma_1)$ to $\text{CSP}(\Gamma_2)$.

Intuition: If the function of a constraint c is pp-definable over Γ through a formula ϕ , then

- Add one fresh variable for each existentially quantified variable in ϕ
- Replace c with constraints from Γ given by the atoms of ϕ

Definition

The **relational clone** of a language Γ , denoted by $\langle \Gamma \rangle$, is the set of all functions pp-definable over Γ .

Intuition: $\langle \Gamma \rangle \approx$ set of all functions that can be “expressed” by Γ

The complexity of $\text{CSP}(\Gamma)$ is tightly connected to the functions in $\langle \Gamma \rangle$:

- Any algorithm that solves $\text{CSP}(\Gamma)$ also solves $\text{CSP}(\Gamma')$ for every $\Gamma' \subset \langle \Gamma \rangle$
- If $\langle \Gamma \rangle$ contains a well-known NP-hard language, then Γ is NP-hard
- If it does not, we can deduce useful structural information about Γ : the existence of non-trivial **polymorphisms**.

Polymorphisms

Given an operation $f : D^k \rightarrow D$ and k tuples $\tau_1, \dots, \tau_k \in D^q$, we write $f(\tau_1, \dots, \tau_k)$ for the tuple $(f(\tau_1[1], \dots, \tau_k[1]), \dots, f(\tau_1[q], \dots, \tau_k[q]))$

Definition

Let Γ be over D . A k -ary **polymorphism** of Γ is an operation $f : D^k \rightarrow D$ such that $\forall c \in \Gamma$ and $\forall \tau_1, \dots, \tau_k \in c$, we have that $f(\tau_1, \dots, \tau_k) \in c$.

Example

$$D = \{0, 1\}, f(a_1, a_2, a_3) = a_1 \oplus a_2 \oplus a_3 = \text{minority}(a_1, a_2, a_3)$$

$c :$
$\begin{array}{ccc} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{array}$
\downarrow
\downarrow
\downarrow

Polymorphisms

Notation: $\text{Pol}(\Gamma) = \text{set of polymorphisms of } \Gamma$

Proposition

Let Γ be a language over a domain D . Then $\text{Pol}(\Gamma)$ is a *concrete clone*:

- (i) $\text{Pol}(\Gamma)$ **contains all projections**, i.e. operations $\pi_i : D^k \rightarrow D$ such that $\pi_i(a_1, \dots, a_m) = a_i$.
- (ii) $\text{Pol}(\Gamma)$ is **closed under composition**, that is, any operation obtained by composing polymorphisms of Γ is also a polymorphism of Γ .

Example: if $f, g, h \in \text{Pol}(\Gamma)$ are of respective arities 3, 2, 1 then the operation $w : D^3 \rightarrow D$ given by

$$w(a_1, a_2, a_3) = f(f(a_1, a_1, h(a_2)), g(a_2, a_3), g(a_1, a_1))$$

is also a polymorphism of Γ .

Geiger's Theorem

Theorem [Geiger 1968]

Let Γ_1, Γ_2 be two constraint languages over the same domain D . Then, $\Gamma_2 \subseteq \langle \Gamma_1 \rangle$ if and only if $\text{Pol}(\Gamma_1) \subseteq \text{Pol}(\Gamma_2)$.

Consequence: the complexity of a language Γ is **completely determined** by its set of polymorphisms

Examples:

- $\text{Pol}(\{c_{\oplus}\})$ contains the operation $f(a) = \bar{a}$, so no Boolean clause is pp-definable over $\{c_{\oplus}\}$
- $\text{Pol}(\{c_{1\text{-in-}3}\})$ only contains projections, so every Boolean function is pp-definable over $\{c_{1\text{-in-}3}\}$

Polymorphisms and complexity

Intuition: If $\text{Pol}(\Gamma)$ contains some highly nontrivial polymorphisms, then $|\Gamma|$ is small, and Γ is unlikely to be NP-hard

Some polymorphisms that guarantee polynomial-time solvability:

- Semilattice polymorphisms: $\forall a, b, c \in D$,

$$f(a, a) = a$$

$$f(a, b) = f(b, a)$$

$$f((f(a, b), c) = f(a, f(b, c))$$

- Majority polymorphisms: $f(a, a, b) = f(a, b, a) = f(b, a, a) = a$
- Minority polymorphisms: $f(a, a, b) = f(a, b, a) = f(b, a, a) = b$
- Mal'tsev polymorphisms: $f(a, a, b) = f(b, a, a) = b$
- Etc.

Schaefer's Theorem

Theorem [Schaefer 1978]

Let Γ be a finite Boolean constraint language. If Γ has one of the following polymorphisms:

- $f(a) = 0$ [Return true]
- $f(a) = 1$ [Return true]
- $f(a_1, a_2) = a_1 \vee a_2$ [Enforce AC]
- $f(a_1, a_2) = a_1 \wedge a_2$ [Enforce AC]
- $f(a_1, a_2, a_3) = \text{majority}(a_1, a_2, a_3)$ [Enforce SAC]
- $f(a_1, a_2, a_3) = \text{minority}(a_1, a_2, a_3)$ [Gaussian elim.]

then $\text{CSP}(\Gamma)$ is polynomial-time. Otherwise, $\text{CSP}(\Gamma)$ is NP-complete.

The Dichotomy Theorem

Theorem [Bulatov 2017][Zhuk 2017]

Let Γ be a finite constraint language. If Γ has a polymorphism such that for all $a, r, e \in D$,

$$f(a, r, e, a) = f(r, a, r, e)$$

then $\text{CSP}(\Gamma)$ is polynomial time. Otherwise, $\text{CSP}(\Gamma)$ is NP-complete.

- Many other equivalent formulations
- There is a dichotomy: $\text{CSP}(\Gamma)$ is either in P or NP-complete
 - ▶ Conjectured in 1993, proved 24 years later
- Generalises Schaefer's Theorem, but much harder to use
 - ▶ Checking if such a polymorphism exists is NP-complete
- The polynomial-time algorithm is extremely impractical

AC-solvable languages

Some polynomial-time languages Γ admit much simpler algorithms, e.g. arc consistency

Theorem [Dalmau, Pearson 1999]

Let Γ be a constraint language. Then, $\text{CSP}(\Gamma)$ is solved by AC if and only if for every $k > 0$ there exists a polymorphism f of arity k that is *totally symmetric*:

$$\text{If } \{a_1, \dots, a_k\} = \{b_1, \dots, b_k\}, \text{ then } f(a_1, \dots, a_k) = f(b_1, \dots, b_k)$$

Example: $\max(a_1, \dots, a_k)$

Bounded width

A constraint language Γ has *bounded width* if there exists a finite k such that strong k -consistency solves $\text{CSP}(\Gamma)$.

A *weak near-unanimity operation* (WNU) is an operation f such that $\forall a, b \in D, f(b, a, a, \dots, a) = f(a, b, a, \dots, a) = \dots = f(a, a, a, \dots, b)$.

Theorem [Barto, Kozik 2009]

Let Γ be a constraint language. Then, $\text{CSP}(\Gamma)$ has bounded width if and only if it has two WNU polymorphisms f, g of arities 3, 4 s.t. $\forall a, b \in D,$

$$f(b, a, a) = g(b, a, a, a)$$

If Γ has bounded width, then SAC solves $\text{CSP}(\Gamma)$! [Kozik 2016]

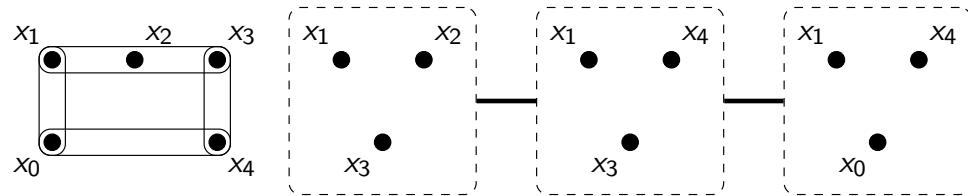
Part 2: structure-based tractable classes

Tree decompositions

Definition: Tree decomposition

A **tree decomposition** of a hypergraph H is a pair $(T, (B_t)_{t \in V(T)})$ such that

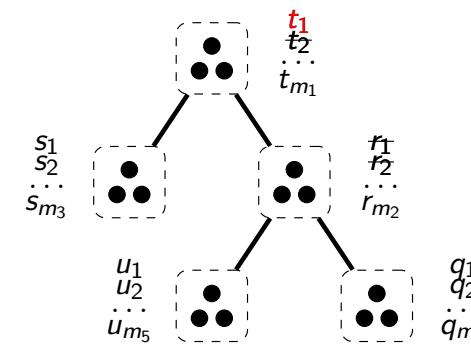
- (i) T is a tree;
- (ii) Each “bag” $B_t, t \in V(T)$ is a set of vertices of H ;
- (iii) Each edge of H is **completely contained** in at least one bag;
- (iv) For each vertex v of H , the bags that contain v form a **connected subtree** of T .



Solving via tree decompositions

Fact

Given a CSP instance I and a tree decomposition of $H(I)$, **if we can enumerate all possible assignments for all bags in polynomial time, then we can solve I in polynomial time.**



1. Compute all consistent assignments to bags
2. Bottom-up: remove assignments that cannot be extended to every child
3. Return YES iff there is at least one assignment left at the root

Solving via tree decompositions

Algorithm Solve($N : \text{CSP}$; $(T, (B_t)_{t \in V(T)})$ tree dec. of $H(N)$):

for each $t \in V(T)$ **from the leaves to the root do**

$S_t \leftarrow \{\tau : B_t \rightarrow D \mid \tau \text{ is consistent}\}$ ¹;

for each $t_c \in V(T) : t_c \text{ is a child of } t$ **do**

$S_t \leftarrow \{\tau \in S_t \mid \exists \tau_c \in S_{t_c} : \tau_c \cup \tau \text{ is consistent}\}$;

$r \leftarrow \text{root}(T)$;

return ($S_r \neq \emptyset$);

Correctness:

- N has a solution $\phi \Rightarrow \phi[B_t] \in S_t$ for all $t \in V(T)$: Solve returns true
- Solve returns true $\Rightarrow \text{root}(T)$ has a consistent assignment $\phi_r \in S_r$
 - $\Rightarrow \phi_r$ can be extended to a complete assignment ϕ by picking arbitrary compatible assignments in each S_t (uses vertex-connectivity)
 - $\Rightarrow \phi$ satisfies all constraints (uses edge containment)

¹here "consistent" means that it satisfies the projection onto B_t of every constraint.

Treewidth

The **width** of $(T, (B_t)_{t \in V(T)})$ is the maximum size of a bag B_t , minus one.

The **treewidth** of a hypergraph H is the minimum width of a tree decomposition of H .

Fact [Freuder 1990]

$\text{CSP}(\mathcal{H}, -)$ is polynomial-time if \mathcal{H} has bounded treewidth.

Proof: Compute an optimal tree decomposition in linear time [Bodlaender 1992] and then use algorithm Solve.

Remarks:

- Simply establishing strong $(k + 1)$ -consistency also works, where k is the treewidth of \mathcal{H}
- Under mild assumptions, if the arity is bounded then $\text{CSP}(\mathcal{H}, -)$ polynomial-time $\iff \mathcal{H}$ has bounded treewidth [Grohe 2006]

Hypertreewidth

Fact

If a subset X' of variables is covered by k constraint with at most t tuples each, then it has at most t^k consistent assignments.

Proof sketch:

Let c_1, \dots, c_k be k covering constraints and $S = \{(\tau_1, \dots, \tau_k) \mid \tau_i \in c_i\}$. Each consistent assignment ϕ to X' satisfies $c_1[X'], \dots, c_k[X']$, so we can map ϕ to at least one element of S . Since c_1, \dots, c_k covers X' , this mapping is injective: there are at most $|S| \leq t^k$ consistent assignments.

Hypertreewidth

The **c-width** of a tree decomposition $(T, (B_t)_{t \in V(T)})$ is the least k such that every bag B_t is k -covered

The (generalised) **hypertreewidth** of a hypergraph H is the minimum c-width of a tree decomposition of H [Gottlob, Leone, Scarcello 1999]

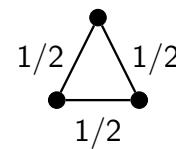
Theorem [Gottlob, Leone, Scarcello 1999]

$\text{CSP}(\mathcal{H}, -)$ is polynomial-time if \mathcal{H} has bounded hypertreewidth.

- Given a fixed $k \geq 2$, computing a tree decomposition of c-width $\leq k$ if one exists is NP-hard [Fischl, Gottlob, Pichler 2018]
- It is however possible to compute a tree dec. of c-width at most $3k$ in polynomial time (or conclude that H has hypertreewidth $> k$)

Bounded fractional hypertreewidth

A **fractional edge cover** of a hypergraph is a weight assignment γ to edges such that for every vertex v , $\sum_{e \in H: v \in E} \gamma(e) \geq 1$.



The AGM bound [Atserias, Grohe, Marx 2008]

If $I = (X, D, C)$ is a CSP instance with hypergraph H_I and γ is a fractional edge cover of H_I of total weight k , then I has at most

$$\prod_{c \in C} |c|^{\gamma(c)} \leq t^k$$

solutions, and this upper bound is tight.

Fractional hypertreewidth

The **fc-width** of $(T, (B_t)_{t \in V(T)})$ is the least k such that every bag B_t has a **fractional edge cover of weight at most k** .

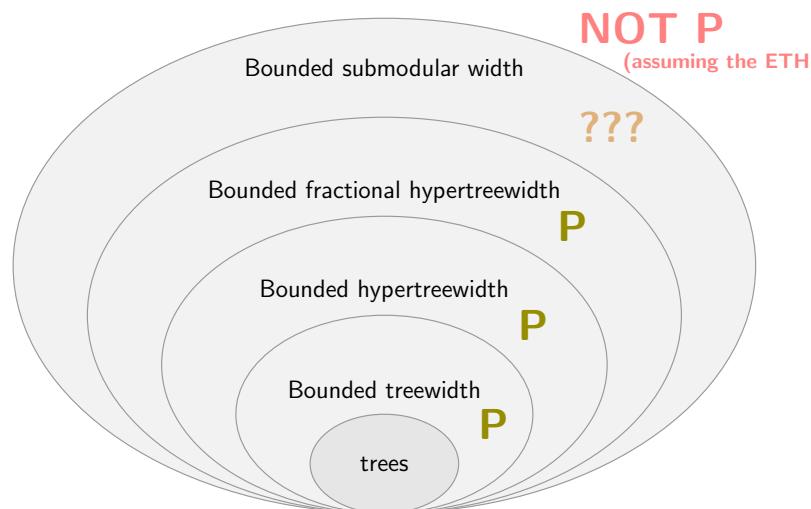
The **fractional hypertreewidth** of H is the minimum fc-width of a tree decomposition of H [Grohe, Marx 2006].

Theorem [Grohe, Marx 2006]

$\text{CSP}(\mathcal{H}, -)$ is polynomial-time if \mathcal{H} has bounded fractional hypertreewidth.

- As for hypertreewidth, given a fixed $k \geq 2$ deciding if there exists a tree decomposition of fc-width $\leq k$ is NP-hard
- An approximately optimal tree decomposition can be computed in polynomial time, but this time the best approximation known is k^3 (instead of $3k$ for hypertreewidth) [Marx 2009]

State of the art: structure-based classes



Qualitative Reasoning: An Overview

Definition

A qualitative constraint language is based on a finite set B of base relations with the following properties:

- the base relations are defined over an infinite domain D
- the base relations are jointly exhaustive & pairwise disjoint (JEPD)
- B contains the identity relation Id
- B is closed under the converse operation ($^{-1}$)
- 2^B is equipped with the usual set-theoretic operations union and intersection, the converse operation, and the weak composition operation (\diamond)

We will look into these notions in detail in the next slides!

- They are *jointly exhaustive and pairwise disjoint* (JEPD):

- $\bigcup\{b \in B\} = \overbrace{D \times \dots \times D}^{\epsilon \text{ times}}$ (jointly exhaustive)
- $\forall b, b' \in B$ such that $b \neq b'$, we have that $b \cap b' = \emptyset$ (pairwise disjoint)
- In other words: A tuple of ϵ elements of D can appear in / satisfy¹¹ at most one single base relation $b \in B$, i.e., each base relation represents a distinct set of tuples

¹¹We will define this term later on

Base Relations: Point Algebra

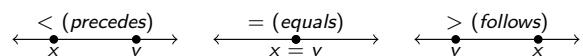


Figure: The 3 base relations of Point Algebra; $>$ is the inverse of $<$

- $D = \mathbb{Q}$ (i.e., the set of rational numbers)
- $B = \{<, = (\text{Id}), >\}$
- $2^B = \{\{<, =, >\}, \{<, >\}, \{<, =\}, \{=, >\}, \{<\}, \{>\}, \{=\}, \emptyset\}$
- Arity $\epsilon = 2$; this will always be the case in this course from now on!

Example

$$\text{precedes} = \{(x, y) \in \mathbb{D}^2 \mid x < y\}^{13}$$

¹³'<' here is a comparison operator and, hence, different from the '<' symbol in the figure, albeit they convey the same semantics (any symbol can be chosen, but we opt for intuitive ones)

Base Relations: Allen's Interval Algebra

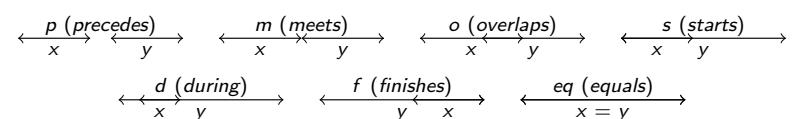


Figure: The 13 base relations of Interval Algebra; inverses are omitted in the figure

- $D = \{x = (x^-, x^+) \in \mathbb{Q}^2 \mid x^- < x^+\}$
- $B = \{eq (\text{Id}), p, pi, m, mi, o, oi, s, si, d, di, f, fi\}$

Example

$$\text{during} = \{(x, y) \in \mathbb{D}^2 \mid x^- > y^- \wedge x^+ < y^+\}$$

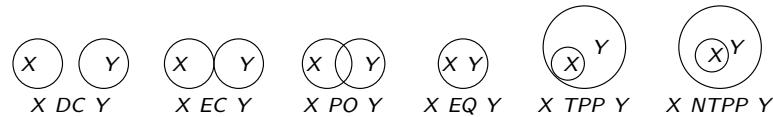


Figure: The 8 base relations of RCC8; inverses are omitted in the figure

- $D = \text{the set } T_{\text{reg}} \text{ of all non-empty regular closed subsets of some topological space } T, \text{ in particular the spaces } \mathbb{R}^n \text{ for some } n \geq 1$
- $B = \{DC, EC, EQ (= \text{Id}), PO, TPP, TPPi, NTPP, NTPPi\}^{14}$

¹⁴Respectively, disconnected, externally connected, equals, partially overlaps, tangential proper part, tangential proper part inverse, non-tangential proper part, non-tangential proper part inverse

Given two base relations $b, b' \in B$, we have:

$$b \diamond b' = \{b'' \in B \mid b'' \cap (b \circ b') \neq \emptyset\}$$

Given two relations $r, r' \in 2^B$, we have:

$$r \diamond r' = \bigcup \{b \diamond b' \mid b \in r, b' \in r'\}$$

Example

$\{EC\} \diamond \{EC\} \supseteq \{EC\}$, and, of course, $\{<, =\} \diamond \{<\} = \{<\}$

Relational Operations: RCC8 Weak Composition Table

\diamond	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ
DC	B	$DC, EC, PO, TPP, NTPP$	$DC, EC, PO, TPP, NTPP$	$DC, EC, PO, TPP, NTPP$	$DC, EC, PO, TPP, NTPP$	DC	DC	DC
EC	$DC, EC, PO, TPPi, NTPPi$	$DC, EC, PO, TPP(i), EQ$	$DC, EC, PO, TPP, NTPP$	$EC, PO, TPP, NTPP$	$PO, TPP, NTPP$	DC, EC	DC	EC
PO	$DC, EC, PO, TPPi, NTPPi$	$DC, EC, PO, TPPi, NTPPi$	B	$PO, TPP, NTPP$	$PO, TPP, NTPP$	$DC, EC, PO, TPPi, NTPPi$	$DC, EC, PO, TPPi, NTPPi$	PO
TPP	DC	DC, EC	$DC, EC, PO, TPP, NTPP$	TPP, NTPP	NTPP	$DC, EC, PO, TPP(i), EQ$	$DC, EC, PO, TPPi, NTPPi$	TPP
NTPP	DC	DC	$DC, EC, PO, TPP, NTPP$	NTPP	NTPP	$DC, EC, PO, TPP, NTPP$	B	NTPP
TPPi	$DC, EC, PO, TPPi, NTPPi$	$EC, PO, TPPi, NTPPi$	$PO, TPPi, NTPPi$	$PO, TPP(i), EQ$	$PO, TPP, NTPP$	$TPPi, NTPPi$	NTPPi	TPPi
NTPPi	$DC, EC, PO, TPPi, NTPPi$	$PO, TPPi, NTPPi$	$PO, TPPi, NTPPi$	$PO, TPP(i), NTPPi, EQ$	NTPPi	NTPPi	NTPPi	NTPPi
EQ	DC	EC	PO	TPP	NTPP	TPPi	NTPPi	EQ

Relation Algebras

Table: Axioms for relation algebras, where $r, s, t \in 2^B$

Axiom	Definition
U-commutativity	$r \cup s = s \cup r$
U-associativity	$r \cup (s \cup t) = (r \cup s) \cup t$
Huntington axiom	$\bar{r} \cup \bar{s} \cup \bar{r} \cup s = r$
\diamond -associativity	$r \diamond (s \diamond t) = (r \diamond s) \diamond t$
\diamond -distributivity	$(r \cup s) \diamond t = (r \diamond t) \cup (s \diamond t)$
identity law	$r \diamond \text{Id} = r$
${}^{-1}$ -involution	$(r^{-1})^{-1} = r$
${}^{-1}$ -distributivity	$(r \cup s)^{-1} = r^{-1} \cup s^{-1}$
${}^{-1}$ -involutive distributivity	$(r \diamond s)^{-1} = s^{-1} \diamond r^{-1}$
Tarski/de Morgan axiom	$r^{-1} \diamond \bar{r} \diamond \bar{s} \cup \bar{s} = \bar{s}$

If a qualitative constraint language satisfies the above axioms, it is a *relation algebra* in the sense of Tarski¹⁶

¹⁶A. Tarski: *On the calculus of relations*. J. Symb. Log. (1941)

Proposition

Each of the following qualitative constraint languages is a relation algebra with the respective algebraic structure $(2^B, \text{Id}, \diamond, {}^{-1})$:

- Point Algebra
- Interval Algebra
- RCC8

Of course, many more calculi exist, and new ones may be constructed, with the above property

- 2^B is by definition closed under weak composition, union, intersection, and converse
- In the context of refinement algorithms that we will see later on, union is not important to us

Definition

A subclass of relations is a subset $\mathcal{A} \subseteq 2^B$ that contains the singleton relations of 2^B and is closed under converse (${}^{-1}$), intersection (\cap), and weak composition (\diamond)

Clearly, the entire set of relations 2^B is also a subclass of relations

Reasoning with Qualitative Constraint Networks**Qualitative Constraint Network: Definition**

Spatial or temporal information for a set of entities can be represented by a qualitative constraint network (QCN)

Definition

A qualitative constraint network (QCN) of some qualitative constraint language is a tuple (V, C) where:

- V is a set of variables over the infinite domain D of the language;
- and C is a mapping $C : V \times V \rightarrow 2^B$ associating a relation (set of base relations) of the language with each pair of variables in V

By definition, a QCN is defined w.r.t some qualitative constraint language, like Point Algebra and so on¹⁸

¹⁸This becomes obvious through the use of the set B

Let us consider a non-normalized QCN where we have the constraints:

$$C(i, j) = \{<, =\} \text{ and } C(j, i) = \{<, =\}$$

This would mean that we want to have:

$i \{<, =\} j$ and $j \{<, =\} i$ (impossible to strictly order the variables)

We can enforce the normalization condition by performing:

$$C(i, j) \leftarrow (C(j, i))^{-1} \cap C(i, j)$$

$$C(j, i) \leftarrow (C(i, j))^{-1} \cap C(j, i)$$

- A QCN $\mathcal{N} = (V, C)$ is *trivially inconsistent* iff $\exists v, v' \in V$ such that $C(v, v') = \emptyset$
- A *solution* of a QCN $\mathcal{N} = (V, C)$ is a mapping $\sigma : V \rightarrow D$ such that, $\forall v, v' \in V, \exists b \in C(v, v')$ such that $(\sigma(v), \sigma(v')) \in b$
- \mathcal{N} is *satisfiable* (or *consistent*)²⁰ if and only if it admits a solution
- A *sub-QCN*²¹ \mathcal{N}' of \mathcal{N} , denoted by $\mathcal{N}' \subseteq \mathcal{N}$, is a QCN (V, C') such that, $\forall u, v \in V, C'(u, v) \subseteq C(u, v)$
- A *scenario* of \mathcal{N} is a consistent atomic sub-QCN \mathcal{S} of \mathcal{N} , where a QCN $\mathcal{S} = (V, C')$ is *atomic* iff, $\forall v, v' \in V, |C'(v, v')| = 1$

²⁰What is the difference between the two terms, if any?

²¹This term can also be found under the name *refined QCN* or *refinement* throughout the literature

Reasoning Problems of QCNs: Minimal Labeling

Definition

Given a QCN $\mathcal{N} = (V, C)$ and a constraint $C(u, v)$ with $u, v \in V$, the minimal labeling problem is deciding if $C(u, v)$ contains unfeasible base relations (i.e., base relations that do not appear in any scenario of \mathcal{N})

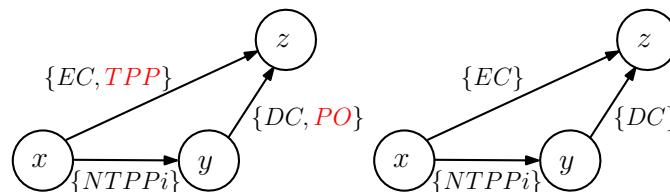


Figure: A RCC8 network (left) and its minimal network (right)

The minimal labeling problem is polynomial-time Turing reducible (Cook reducible) to the satisfiability checking problem

Reasoning Problems of QCNs: Redundancy

Definition

Given a QCN $\mathcal{N} = (V, C)$ and a constraint $C(u, v)$ with $u, v \in V$, the redundancy problem is deciding if $C(u, v)$ is entailed by the rest of the constraints of \mathcal{N} (i.e., it is redundant in \mathcal{N})

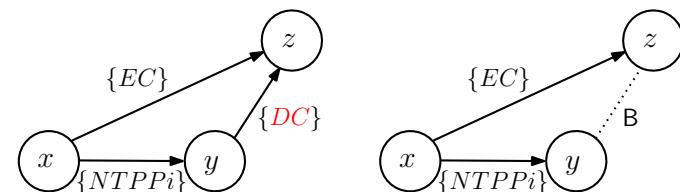


Figure: A RCC8 network (left) and its prime network (right)

Similarly to the minimal labeling problem, the redundancy problem is polynomial-time Turing reducible to the satisfiability checking problem

Definition

Given a QCN $\mathcal{N} = (V, C)$ and a graph $G = (V, E)$, \mathcal{N} is \diamond_G -consistent iff, $\forall \{v_i, v_j\}, \{v_i, v_k\}, \{v_k, v_j\} \in E, C(v_i, v_j) \subseteq C(v_i, v_k) \diamond C(v_k, v_j)$

Intuitively, \diamond_G -consistency checks if all triples of variables in \mathcal{N} that correspond to triangles in G are closed under weak composition²⁴

For each of the qualitative constraint languages of Point Algebra, Interval Algebra, and RCC8²⁵ we have:

Property

Every \diamond -consistent atomic QCN of Point Algebra, Interval Algebra, or RCC8 is satisfiable

In fact, later on we will see that the above result holds for \diamond_G -consistency too, when G satisfies certain criteria other than being complete

²⁴M. Sioutis et al.: An Efficient Approach for Tackling Large Real World Qualitative Spatial Networks. Int. J. Artif. Intell. Tools 25 (2016)

²⁵Many more calculi exist with this property, but these are the ones we focus on in this course

\diamond_G -Consistency: Complexity of Checking

Given a QCN $\mathcal{N} = (V, C)$ and a graph $G = (V, E)$, we need to check if:

$$\forall \{v_i, v_j\}, \{v_i, v_k\}, \{v_k, v_j\} \in E, C(v_i, v_j) \subseteq C(v_i, v_k) \diamond C(v_k, v_j)$$

- Basically, and as mentioned earlier, we need to visit all triangles in G
- Thus, runtime is: $O(\Delta \cdot |E|)$, where Δ is the maximum degree of G
- If G is a complete graph, we get $O(|V|^3)$ (why?)

Algebraic Closure: SOTA Algorithm (2/3)

Algorithm 1: PWC(\mathcal{N}, G)

```

in      : A QCN  $\mathcal{N} = (V, C)$  and a graph  $G = (V, E)$ .
output   : A sub-QCN of  $\mathcal{N}$ .
begin
1    $Q \leftarrow E;$ 
2   while  $Q \neq \emptyset$  do
3      $\{v, v'\} \leftarrow Q.pop();$ 
4     foreach  $v'' \in V \mid \{v, v''\}, \{v', v''\} \in E$  do
5        $r \leftarrow C(v, v'') \cap (C(v, v') \diamond C(v', v''));$ 
6       if  $r \subset C(v, v'')$  then
7          $C(v, v'') \leftarrow r;$ 
8          $C(v'', v) \leftarrow r^{-1};$ 
9          $Q \leftarrow Q \cup \{\{v, v''\}\};$ 
10     $r \leftarrow C(v'', v') \cap (C(v'', v) \diamond C(v, v'));$ 
11    if  $r \subset C(v'', v')$  then
12       $C(v'', v') \leftarrow r;$ 
13       $C(v', v'') \leftarrow r^{-1};$ 
14       $Q \leftarrow Q \cup \{\{v'', v'\}\};$ 
15
16  return  $\mathcal{N};$ 

```

- Naive approach: $O(\Delta \cdot |E|^2 \cdot |B|)$ time²⁸

- SOTA approach: $O(\Delta \cdot |E| \cdot |B|)$ time²⁹

- What about space complexity?

- Naive approach: $O(1)$ (why?)

- SOTA approach: $O(|E|)$ (maintained by a queue)

- What about best case complexity? $\Omega(\Delta \cdot |E|)$ for both

$\diamond_G(\mathcal{N})$ denotes the algebraic closure of \mathcal{N} under \diamond_G -consistency

Property (Soundness)

Given a QCN $\mathcal{N} = (V, C)$ and a graph $G = (V, E)$, if $\emptyset \in \diamond_G(\mathcal{N})$ (i.e., if $\diamond_G(\mathcal{N})$ is trivially inconsistent), then \mathcal{N} is unsatisfiable

In general, the algebraic closure is NOT complete for deciding satisfiability of a QCN!³⁰

²⁸ $O(|V|^5 \cdot |B|)$ time when G is complete

²⁹ $O(|V|^3 \cdot |B|)$ time when G is complete

³⁰ Unless, with what we have seen so far, the QCN is refined to an \diamond -consistent atomic sub-QCN; later this result will be generalized to certain non-atomic QCNs too

Algebraic Closure: Properties w.r.t \diamond_G -Consistency

- $\diamond_G(\mathcal{N})$ is the largest³¹ \diamond_G -consistent sub-QCN of \mathcal{N} (Dominance)

- $\diamond_G(\mathcal{N})$ is equivalent³² to \mathcal{N} (Equivalence)

- $\diamond_G(\diamond_G(\mathcal{N})) = \diamond_G(\mathcal{N})$ (Idempotence)

- if $\mathcal{N}' \subseteq \mathcal{N}$ then $\diamond_G(\mathcal{N}') \subseteq \diamond_G(\mathcal{N})$ (Monotonicity)

Tractable Subclasses of Relations: Definition

Reminder of subclass of relations: A subclass of relations is a subset $\mathcal{A} \subseteq 2^B$ that contains the singleton relations of 2^B and B and is closed under converse ($^{-1}$), intersection (\cap), and weak composition (\diamond)

Definition

A subclass of relations \mathcal{A} is tractable iff the class of QCNs defined over \mathcal{A} is tractable, i.e., the satisfiability of every QCN in that class can be decided in polynomial time

In this course, we focus on tractability w.r.t \diamond_G -consistency³⁴

³¹ W.r.t \subseteq

³² Two QCNs are equivalent if they have the same set of solutions

³⁴ Indeed, other polytime methods may exist for deciding the satisfiability of a QCN

A Horn theory of Interval Algebra can be based on that of partial orders:³⁶

$$\begin{array}{ll} x \leq z \wedge z \leq y \rightarrow x \leq y & x = y \rightarrow x \leq y \\ x \leq y \wedge y \leq x \rightarrow x = y & x = y \rightarrow y \leq x \\ x = y \wedge x \neq y \rightarrow \perp & x \neq x \rightarrow \perp \end{array}$$

Then, every interval variable $x = (x^-, x^+)$ in a QCN can be translated as follows (remember that $x^- < x^+$):

$$x^- \leq x^+ \wedge x^- \neq x^+$$

In addition, for all distinct interval variables x , y , and z in our QCN, and all their endpoints, we need to enforce the theory of partial orders above

- Given a QCN, we can obtain a CNF formula for each of its constraints
- The formula is Horn if it contains only clauses with:
 - *at most one* positive literal, i.e., of the form $x = y$ or $x \leq y$
 - and an arbitrary number of negative literals, i.e., of the form $x \neq y$
- If all the formulas (constraints) are Horn, the QCN is tractable (because of Horn-satisfiability)

³⁶Bernhard Nebel, Hans-Jürgen Bürckert: *Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra*. J. ACM 42 (1974)

Tractable Subclasses of Relations: Maximality

Definition

A tractable subclass of relations \mathcal{A} is maximal iff there is not other tractable subclass that properly/strictly contains \mathcal{A}

- Once a tractability result is known/proved, other (large) tractable classes may be identified automatically³⁸
- However, *maximality* of a tractable subclass requires formal theoretical analysis

Tractable Subclasses of Relations: Distributivity

Given three relations $r, s, t \in 2^B$, weak composition distributes over intersection if we have that $r \diamond (s \cap t) = (r \diamond s) \cap (r \diamond t)$ and $(s \cap t) \diamond r = (s \diamond r) \cap (t \diamond r)$

Definition

A tractable subclass of relations \mathcal{A} is *distributive* iff weak composition distributes over non-empty intersection $\forall r, s, t \in \mathcal{A}$

Distributive subclasses of relations exhibit *convexity* in Helly's sense³⁹

³⁸J. Renz: *Qualitative Spatial and Temporal Reasoning: Efficient Algorithms for Everyone*. In: IJCAI 2007

³⁹L. Danzer et al.: *Helly's Theorem and Its Relatives*. Proceedings of symposia in pure mathematics: Convexity 7 (1963)

Definition

A subclass of relations $\mathcal{A} \subseteq 2^B$ is Helly if and only if for any n relations $r_1, r_2, \dots, r_n \in \mathcal{A}$ we have:

$$\bigcap_{i=1}^n r_i \neq \emptyset \text{ iff, } \forall i, j \in \{1, \dots, n\}, r_i \cap r_j \neq \emptyset$$

Then, we have the following result by Long and Li:⁴⁰

Theorem

A subclass of relations $\mathcal{A} \subseteq 2^B$ is distributive if and only if it is Helly

⁴⁰Z. Long and S. Li: *On Distributive Subalgebras of Qualitative Spatial and Temporal Calculi*.
In: COSIT 2015

A tractable subclass of relations that is also distributive, makes the effect of \diamond_G -consistency stronger

Theorem

If every \diamond -consistent atomic QCN of a qualitative constraint language that is based on a set of base relations B is satisfiable, then every \diamond -consistent QCN defined over a distributive subclass of relations $\mathcal{A} \subseteq 2^B$ is minimal, i.e., it contains no unfeasible base relations

Clearly, this result holds for Point Algebra, Interval Algebra, and RCC8

Tractable Subclasses of Relations: Patchwork Property

Given two QCNs $\mathcal{N} = (V, C)$ and $\mathcal{N}' = (V', C')$ with identical labeling of constraints on $V \cap V'$,⁴² $\mathcal{N} \sqcup \mathcal{N}'$ denotes the QCN (V'', C'') where:

- $V'' = V \cup V'$
- $\forall (u, v) \in (V \setminus V') \times (V' \setminus V), C''(u, v) = C''(v, u) = B$
- $\forall u, v \in V, C''(u, v) = C(u, v)$
- $\forall u, v \in V', C''(u, v) = C'(u, v)$

Theorem

A tractable subclass of relations $\mathcal{A} \subseteq 2^B$ has patchwork if and only if for every \diamond -consistent and not trivially inconsistent QCNs $\mathcal{N} = (V, C)$ and $\mathcal{N}' = (V', C')$ with identical labeling of constraints on $V \cap V'$, the QCN $\mathcal{N} \sqcup \mathcal{N}'$ is a satisfiable

⁴²Formally, $\forall u, v \in V \cap V', C(u, v) = C'(u, v)$

Patchwork Property: Revisiting \diamond_G -Consistency

We use $G(\mathcal{N})$ to denote the *constraint graph*⁴⁴ (V, E) of a QCN $\mathcal{N} = (V, C)$, where $\{v, v'\} \in E$ iff $C(v, v') \neq B$

Proposition

Given a tractable subclass of relations $\mathcal{A} \subseteq 2^B$ with patchwork and a QCN \mathcal{N} defined over \mathcal{A} , if \mathcal{N} is \diamond_G -consistent and not trivially inconsistent, where G is a chordal graph such that $G(\mathcal{N}) \subseteq G$, then \mathcal{N} is satisfiable

But what is a chordal graph and how does it play a role here?

⁴⁴Also called *primal* constraint graph in the literature

Given a QCN $\mathcal{N} = (V, C)$, a chordal graph $G = (V, E) \supseteq G(\mathcal{N})$, and a subclass \mathcal{A} for which ${}^\diamond_G$ -consistency is complete, we aim to characterize a QCN \mathcal{N}' such that:

$$\mathcal{N}' \subseteq \mathcal{N} \text{ and } \mathcal{N}' = {}^\diamond_G(\mathcal{A}(\mathcal{N}'))$$

Let us consider a QCN \mathcal{N} , we tackle it as follows:

- Every relation r forming a constraint in \mathcal{N} is split into subrelations $r_i \subseteq r$
- These subrelations r_i belong to a set of relations \mathcal{A} over which the QCN becomes tractable
- After every refinement of a relation r into one of its subrelations r_i , a validity check is performed:
 - if the refinement is valid, we proceed with the next one;
 - if the refinement is not valid, we backtrack to the previous one
- When \mathcal{N} has been refined into relations of \mathcal{A} , we stop

From Local Consistency to Satisfiability: Algorithm (1/3)

Algorithm 2: Consistency($\mathcal{N}, G, \mathcal{A}$)

```

in      : A QCN  $\mathcal{N} = (V, C)$ , a graph  $G = (V, E)$ , and a subclass  $\mathcal{A}$  of  $2^B$ .
output   : Null, or a refinement of network  $\mathcal{N}$  over  $\mathcal{A}$ .
1 begin
2    $\mathcal{N}' \leftarrow \text{PWC}(\mathcal{N}, G);$ 
3   if  $\exists \{v_i, v_j\} \in E$  such that  $C(v_i, v_j) = \emptyset$  then
4     return Null;
5   if  $\forall \{v_i, v_j\} \in E$  we have that  $C(v_i, v_j) \in \mathcal{A}$  then
6     return  $\mathcal{N}'$ ;
7   choose a constraint  $C(v_i, v_j)$  with  $\{v_i, v_j\} \in E$  such that  $C(v_i, v_j) \notin \mathcal{A}$ ;
8   split  $C(v_i, v_j)$  into  $r_1, \dots, r_k \in \mathcal{A}$ :  $r_1 \cup \dots \cup r_k = C(v_i, v_j)$ ;
9   foreach  $r \in \{r_l \mid 1 \leq l \leq k\}$  do
10     $\mathcal{N}'[v_i, v_j] \leftarrow r$ ;  $\mathcal{N}'[v_j, v_i] \leftarrow r^{-1}$ ;
11    result  $\leftarrow \text{Consistency}(\mathcal{N}', G, \mathcal{A})$ ;
12    if result  $\neq$  Null then
13      return result;
14

```

Université de Montpellier - Master 2
Module **Contraintes**

Feuille TD 1 - 18/09/2023

Exercice 1

On considère les réseaux de contraintes suivants :

- $N_1 = (X, D, C)$, où $X = \{X_1, X_2, X_3\}$, $D(X_1) = \{1, 2, 3, 4, 5\}$, $D(X_2) = \{1, 3, 5\}$, $D(X_3) = \{1, 3, 4, 7\}$, et

$$C = \begin{cases} X_1 + X_2 = X_3 \\ X_1 < X_2 \end{cases}$$

- $N_2 = (X, D, C)$, où $X = \{X_1, \dots, X_4\}$, $\forall i D(X_i) = \{1, \dots, 8\}$, et

$$C = \begin{cases} 2 \cdot X_1 = X_3 \\ X_1 < X_2 \\ X_3 + X_4 = 10 \\ X_3 < X_2 \end{cases}$$

- $N_3 = (X, D, C)$, où $X = \{X_1, \dots, X_5\}$, $D(X_1) = D(X_2) = \{1, 2, 7, 8\}$, $D(X_3) = D(X_4) = \{1, 2, 3, 4\}$, $D(X_5) = \{0, 1\}$ et

$$C = \begin{cases} X_1 + X_4 = 10 - 5 \cdot X_5 \\ |X_1 - X_2| = |X_3 - X_4| \\ X_3 = 2 \cdot X_2 \end{cases}$$

Question 1. Appliquez AC sur N_1 .

Correction. On examine les contraintes une par une, en supprimant des valeurs non viables jusqu'à ce que toutes le soient. Attention, il est parfois nécessaire de revenir sur une contrainte déjà examinée car une valeur qui était viable ne l'est plus !

Notation pour cet exercice: " $c_i : (X_j, v)$ " signifie "on supprime la valeur $v \in D(X_j)$ car elle n'a pas de support dans la contrainte c_i ".

$$\begin{aligned} c_2 : (X_1, 5), (X_2, 1) \\ c_1 : (X_1, 3), (X_3, 1), (X_3, 3) \end{aligned}$$

Toutes les valeurs restantes sont viables. La fermeture AC de D est donc :

$$\begin{aligned} D(X_1) &= \{1, 2, 4\} \\ D(X_2) &= \{3, 5\} \\ D(X_3) &= \{4, 7\} \end{aligned}$$

Question 2. Appliquez AC sur N_2 .

Correction.

$$\begin{aligned} c_1 : (X_1, 5), (X_1, 6), (X_1, 7), (X_1, 8), (X_3, 1), (X_3, 3), (X_3, 5), (X_3, 7) \\ c_4 : (X_2, 1), (X_2, 2), (X_3, 8) \\ c_3 : (X_4, 1), (X_4, 2), (X_4, 3), (X_4, 5), (X_4, 7) \\ c_1 : (X_1, 4) \end{aligned}$$

Toutes les valeurs restantes sont viables. La fermeture AC de D est donc :

$$\begin{aligned} D(X_1) &= \{1, 2, 3\} \\ D(X_2) &= \{3, 4, 5, 6, 7, 8\} \\ D(X_3) &= \{2, 4, 6\} \\ D(X_4) &= \{4, 6, 8\} \end{aligned}$$

Question 3. Appliquez AC sur N_3 .

Correction.

$$\begin{aligned} c_3 : (X_2, 7), (X_2, 8), (X_3, 1), (X_3, 3) \\ c_2 : (X_1, 7), (X_1, 8) \\ c_1 : (X_5, 0), (X_4, 1), (X_4, 2) \end{aligned}$$

Toutes les valeurs restantes sont viables. La fermeture AC de D est donc :

$$\begin{aligned} D(X_1) &= \{1, 2\} \\ D(X_2) &= \{1, 2\} \\ D(X_3) &= \{2, 4\} \\ D(X_4) &= \{3, 4\} \\ D(X_5) &= \{1\} \end{aligned}$$

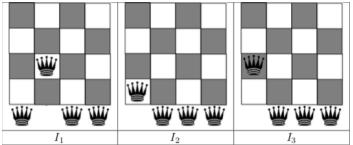
Exercice 2

Le problème des n -reines est de placer n reines de jeu d'échecs sur un échiquier de dimension $n \times n$ sans que les reines ne puissent se menacer mutuellement, conformément aux règles du jeu d'échecs. Par conséquent, deux reines ne peuvent pas se trouver sur la même ligne, colonne, ou diagonale.

On considère un modèle avec n variables X_1, \dots, X_n dont les domaines sont $\{1, \dots, n\}$, où l'affectation $X_i \leftarrow j$ signifie "La reine placée dans la i ème colonne se trouve à la j ème ligne". Les contraintes sont alors:

$$\begin{aligned} \forall i, j \text{ tels que } i \neq j, \quad X_i \neq X_j && \text{(Pas deux reines sur la même ligne)} \\ \forall i, j \text{ tels que } i \neq j, \quad |X_i - X_j| \neq |i - j| && \text{(Pas deux reines sur la même diagonale)} \end{aligned}$$

Pour les questions suivantes, on fixe $n = 4$ et on considère les trois affectations partielles suivantes :



Question 1. Appliquez AC sur le modèle pour chacune des 3 affectations partielles.

Correction.

I₁. Cette affectation partielle impose $D(X_2) = \{3\}$. En réutilisant les notations du premier exercice :

- Contraintes de lignes : $(X_1, 3), (X_3, 3), (X_4, 3)$
- Contraintes de diagonales : $(X_1, 2), (X_1, 4), (X_3, 2), (X_3, 4), (X_4, 1)$
- Contrainte de lignes $X_1 \neq X_3$: $(X_1, 1), (X_3, 1)$

Les domaines de X_1 et X_3 sont désormais vides. AC va ensuite vider tous les autres domaines car les contraintes impliquant l'une ou l'autre de ces variables n'ont plus de support pour aucune valeur.

Après AC : $\forall i, D(X_i) = \emptyset$

I₂. Cette affectation partielle impose $D(X_1) = \{4\}$.

- Contraintes de lignes : $(X_2, 4), (X_3, 4), (X_4, 4)$
- Contraintes de diagonales : $(X_2, 3), (X_3, 2), (X_4, 1)$
- Contrainte de diagonales $|X_2 - X_3| \neq 1$: $(X_2, 2)$
- Contrainte de diagonales $|X_4 - X_3| \neq 1$: $(X_4, 2)$
- Contraintes de lignes $X_2 \neq X_3$ et $X_3 \neq X_4$: $(X_3, 1), (X_3, 3)$

Le domaine de X_3 est désormais vide. **Après AC** : $\forall i, D(X_i) = \emptyset$

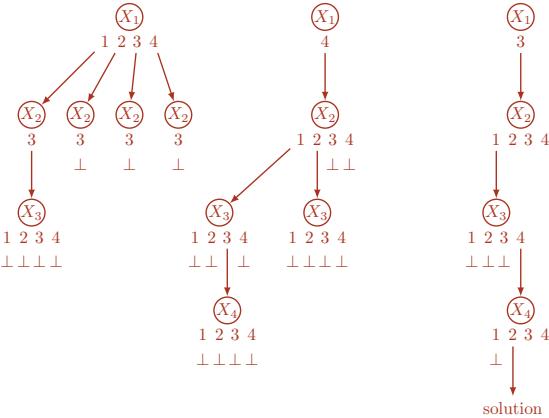
I₃. Cette affectation partielle impose $D(X_1) = \{3\}$.

- Contraintes de lignes : $(X_2, 3), (X_3, 3), (X_4, 3)$
- Contraintes de diagonales : $(X_2, 2), (X_2, 4), (X_3, 1)$
- Contrainte de diagonales $|X_2 - X_3| \neq 1$: $(X_3, 2)$
- Contrainte de lignes $X_2 \neq X_4$: $(X_4, 1)$
- Contrainte de lignes $X_3 \neq X_4$: $(X_4, 4)$

Les domaines sont alors arc cohérents. **Après AC** : $D(X_1) = \{3\}, D(X_2) = \{1\}, D(X_3) = \{4\}, D(X_4) = \{2\}$

Question 2. Appliquez BT sur le modèle pour chacune des 3 affectations partielles, en utilisant pour ordre de branchement de variables (X_1, X_2, X_3, X_4) et ordre de branchement de valeurs $(1, 2, 3, 4)$.

Correction. Voici les arbres explorés par BT sur les 3 affectations partielles (de gauche à droite I_1, I_2 et I_3) :



Question 3. Existe-t-il une affectation partielle qui ne s'étend à aucune solution, mais telle qu'appliquer AC sur le modèle ne vide aucun domaine ? Justifiez votre réponse.

Correction. Supposons par l'absurde qu'une telle affectation partielle existe. Au moins une variable est affectée car sinon l'affectation s'étendrait à une solution. Soit X_i une variable affectée et j sa valeur.

- Premier cas : $i \in \{1, 4\}$. Si $i = 1$, alors d'après la question 2 sur I_2 et I_3 , j ne peut pas valoir 3 ou 4 car sinon AC viderait les domaines ou les réduirait à un singleton. Comme le modèle est symétrique par inversion des lignes, c'est également vrai pour $j \in \{1, 2\}$. De même, comme le modèle est symétrique par inversion des colonnes, c'est également vrai pour $i = 4$. Ce premier cas est donc impossible.
- Deuxième cas : $i \in \{2, 3\}$ et $j \in \{2, 3\}$. D'après la question 2 sur I_1 et par symétrie de lignes/colonnes, dans ce cas AC vide les domaines. Ce cas est donc également impossible.
- Dernier cas : $i \in \{2, 3\}$ et $j \in \{1, 4\}$. Ce cas ne peut pas être déduit directement de la réponse à la question 2. Cependant, on peut aisément vérifier que dans ce cas AC réduit tous les domaines à un singleton lorsque X_i est affecté à j (par symétrie, il suffit de le vérifier pour une seule paire i, j). Ce dernier cas est donc impossible : une telle affectation partielle n'existe donc pas.

Exercice 3

Le graphe primal d'un réseau de contraintes N est le graphe G_N dont les sommets sont les variables de N et les arêtes sont les paires de variables qui apparaissent ensemble dans la portée d'au moins une contrainte. Un réseau de contraintes N est arborescent si et seulement si il est normalisé (toutes les contraintes ont des portées distinctes deux-à-deux) et G_N est un graphe acyclique.

Question 1. Démontrez que tout réseau de contraintes arborescent arc cohérent admet une solution.

Correction. On procède par l'absurde. Supposons qu'il existe un réseau de contraintes arborescent qui est arc cohérent mais n'admet pas de solution. Soit $N = (X, D, C)$ un tel réseau avec $|X|$ de taille minimum.

Premièrement, observons que $|X| > 1$ (sinon N admettrait une solution) et G_N est connexe (sinon N ne serait pas minimal). Il existe donc deux variables $x, y \in X$ telles que x est une feuille de G_N et y est son unique voisin dans G_N . On note $c(x, y)$ l'unique contrainte de N dont la portée contient x et y . Le réseau $N^* = (X \setminus \{x\}, D \setminus \{D(x)\}, C \setminus \{c(x, y)\})$ est arborescent, arc cohérent et contient strictement moins de variables que N ; il admet donc une solution ϕ . La valeur $\phi(y)$ est viable dans N , donc il existe un support $(v, \phi(y)) \in c(x, y)$ avec $v \in D(x)$. Finalement, l'affectation $\phi \cup \{X \leftarrow v\}$ satisfait toutes les contraintes de N , ce qui contredit notre hypothèse initiale.

Question 2. La même propriété est-elle vraie si G_N est un arbre mais N n'est pas normalisé ? Justifiez votre réponse.

Correction. Non. Par exemple, le réseau

$$\begin{aligned} X_1 &= X_2 \\ X_1 &\neq X_2 \end{aligned}$$

avec domaines $D(X_1) = D(X_2) = \{1, 2\}$ est arc cohérent et son graphe primal est acyclique, mais il n'admet pas de solution.

Question 3. Tout réseau de contraintes arborescent arc cohérent est-il globalement cohérent ? Justifiez votre réponse.

Correction. Non. Par exemple, le réseau

$$\begin{aligned} X_1 &= X_2 \\ X_2 &= X_3 \end{aligned}$$

avec domaines $D(X_1) = D(X_2) = D(X_3) = \{1, 2\}$ est arc cohérent et arborescent, mais l'affectation localement cohérente $X_1 \leftarrow 1, X_3 \leftarrow 2$ ne s'étend à aucune solution.

Question 4. L'algorithme BT appliqué à un réseau de contraintes arborescent arc cohérent atteint-il toujours une solution après exploration d'un nombre polynomial d'appels récursifs, indépendamment des choix de branchement ? Justifiez votre réponse.

Correction. Non. Soit n un entier naturel pair et $N = (X, D, C)$, où $X = \{X_1, \dots, X_n\}$, $D(X_i) = \{0, 1\}$ pour tout $i \leq n$ et $C = \{X_1 \neq X_2, X_2 \neq X_3, \dots, X_{n-1} \neq X_n\}$. Considérons un ordre sur les variables qui sélectionne en premier X_1 , puis X_n , puis les variables X_i avec $1 < i < n - 1$ impair dans l'ordre croissant, et enfin les variables restantes. Pour les valeurs, on sélectionne 1 en priorité.

Par construction, l'affectation des $n/2$ premières variables $X_1, X_n, X_3, X_5, \dots, X_{n-3}$ est localement cohérente. Cependant, l'affectation initiale des deux premières variables $X_1 \leftarrow 1, X_n \leftarrow 1$ ne s'étend à aucune solution. L'algorithme BT utilisant ces choix de branchements sur N explorera donc au moins $2^{n/2-2}$ noeuds de l'arbre de recherche avant de trouver une solution.

Exercice 4

Soit le réseau de contraintes $N = (X, D, C)$, où $X = \{X_1, X_2, X_3, X_4\}$, $D(X_1) = \{0, 4, 8, 10\}$, $D(X_2) = D(X_3) = \{0, 1, 2, 5, 8\}$, $D(X_4) = \{0, 3\}$, $c_1 \equiv X_1 = X_2 + X_3$, $c_2 \equiv |X_2 - X_4| \geq 3$, $c_3 \equiv |X_3 - X_4| \leq 2$, et $C = \{c_1, c_2, c_3\}$.

Étant donné un ordre total o sur C , l'algorithme **onepassAC**(o) sélectionne les contraintes de C l'une après l'autre, de la première à la dernière, dans l'ordre o . Une fois une contrainte c sélectionnée, **onepassAC**(o)

supprime toutes les valeurs qui n'ont pas de support sur c puis passe à la contrainte suivante. L'algorithme s'arrête une fois la dernière contrainte traitée.

Question 1. Appliquez **onepassAC**(o) sur N , où $o = (c_1, c_2, c_3)$.

Correction. On utilise les mêmes notations que pour l'exercice 1.

$$\begin{aligned} c_1 &: (X_2, 1), (X_3, 1) \\ c_2 &: (X_2, 2) \\ c_3 &: (X_3, 8) \end{aligned}$$

Après application de **onepassAC**(o), les domaines sont donc

$$\begin{aligned} D(X_1) &= \{0, 4, 8, 10\} \\ D(X_2) &= \{0, 5, 8\} \\ D(X_3) &= \{0, 2, 5\} \\ D(X_4) &= \{0, 3\} \end{aligned}$$

Question 2. Appliquez AC sur N . **onepassAC**(o) est-il en général un algorithme d'arc cohérence ? Justifiez votre réponse.

Correction. On applique maintenant AC. Il est suffisant de partir des domaines obtenus après application de **onepassAC**(o) car les valeurs supprimées ne sont pas viables.

$$c_1 : (X_1, 4)$$

Toutes les valeurs restantes sont viables. La fermeture AC de D est donc :

$$\begin{aligned} D(X_1) &= \{0, 8, 10\} \\ D(X_2) &= \{0, 5, 8\} \\ D(X_3) &= \{0, 2, 5\} \\ D(X_4) &= \{0, 3\} \end{aligned}$$

onepassAC(o) n'est donc pas un algorithme d'arc cohérence car il existe un réseau (en l'occurrence, N) pour lequel les domaines en fin d'exécution ne correspondent pas à la fermeture AC.

Question 3. Étant donné un ordre total o sur C , l'algorithme **doubleAC**(o) exécute d'abord **onepassAC**(o) puis **onepassAC**(o^{-1}), où o^{-1} est l'ordre inverse de o . **doubleAC**(o) est-il en général un algorithme d'arc cohérence ? Justifiez votre réponse.

Correction. Non. Un exemple très simple est le réseau avec deux variables X_1, X_2 , deux contraintes $c_1 \equiv X_1 > X_2$, $c_2 \equiv X_2 > X_1$ et des domaines $D(X_1) = D(X_2) = \{1, 2, 3, 4, 5, 6\}$. En appliquant **doubleAC**(o) pour l'ordre $o = (c_1, c_2)$ on obtient

$$\begin{aligned} c_1 &: (X_1, 1), (X_2, 6) \\ c_2 &: (X_2, 1), (X_2, 2), (X_1, 5), (X_1, 6) \\ c_2 &: \emptyset \\ c_1 &: (X_1, 2), (X_1, 3), (X_2, 4), (X_2, 5) \end{aligned}$$

ce qui donne pour domaine

$$\begin{aligned} D(X_1) &= \{4\} \\ D(X_2) &= \{3\} \end{aligned}$$

qui n'est clairement pas arc cohérent car aucune de ces valeurs n'a de support pour la contrainte c_2 .

Exercice 5

On considère un ensemble de variables $X = \{X_1, X_2, X_3, X_4\}$ dont le domaine D est donné par $D(X_1) = \{1, 2, 3, 4, 5, 6\}$, $D(X_2) = \{1, 2, 4, 5, 6, 7\}$, $D(X_3) = \{1, 2, 3, 4, 5, 6, 7\}$ et $D(X_4) = \{1, 2, 3, 5, 6, 7\}$. Soient les réseaux de contraintes $N_1 = (X, D, C_1)$ et $N_2 = (X, D, C_2)$, où

$$C_1 = \begin{cases} X_1 \leq X_2 \\ X_2 \leq X_3 \\ X_3 \leq X_4 \\ X_4 \leq X_1 \\ X_2 \cdot X_4 \mod 4 = 0 \end{cases} \quad C_2 = \begin{cases} X_1 = X_2 \\ X_2 = X_3 \\ X_3 = X_4 \\ X_4 = X_1 \\ X_2 \cdot X_4 \mod 4 = 0 \end{cases}$$

Question 1. Comparez les ensembles de solutions de N_1 et N_2 . Sont-ils égaux, strictement inclus l'un dans l'autre, disjoints ?

Correction. Les ensembles de solutions de N_1 et N_2 sont égaux car les 4 premières contraintes de C_1 impliquent l'égalité entre les variables X_1, \dots, X_4 .

Question 2. Appliquez AC sur N_1 puis N_2 .

Correction. On note les contraintes c_1, \dots, c_5 (de haut en bas). Pour N_1 , avec les notations du premier exercice :

$$\begin{aligned} c_5 &: (X_2, 1), (X_2, 5), (X_2, 7) \\ c_2 &: (X_3, 1) \\ c_3 &: (X_4, 1) \\ c_4 &: (X_1, 1), (X_4, 7) \\ c_3 &: (X_3, 7) \end{aligned}$$

Les valeurs restantes sont viables. Après AC sur N_1 , les domaines sont donc

$$\begin{aligned} D(X_1) &= \{2, 3, 4, 5, 6\} \\ D(X_2) &= \{2, 4, 6\} \\ D(X_3) &= \{2, 3, 4, 5, 6\} \\ D(X_4) &= \{2, 3, 5, 6\} \end{aligned}$$

On applique maintenant AC sur N_2 :

$$\begin{aligned} c_5 &: (X_2, 1), (X_2, 5), (X_2, 7) \\ c_4 &: (X_1, 4), (X_4, 7) \\ c_1 &: (X_1, 1), (X_1, 3), (X_1, 5), (X_2, 4) \\ c_2 &: (X_3, 1), (X_3, 3), (X_3, 4), (X_3, 5), (X_3, 7) \\ c_3 &: (X_4, 1), (X_4, 3), (X_4, 5) \end{aligned}$$

Les valeurs restantes sont viables. Après AC sur N_2 , les domaines sont donc

$$\begin{aligned} D(X_1) &= \{2, 6\} \\ D(X_2) &= \{2, 6\} \\ D(X_3) &= \{2, 6\} \\ D(X_4) &= \{2, 6\} \end{aligned}$$

Soient $N = (X, D, C)$ et $N^* = (X, D, C^*)$ deux réseaux de contraintes et D_N^{AC} et $D_{N^*}^{AC}$ leurs fermetures arc cohérentes respectives.

Question 3. Supposons que toute solution de N^* est solution de N . Est-il forcément vrai que pour tout $X_i \in X$, on a $D_{N^*}^{AC}(X_i) \subseteq D_N^{AC}(X_i)$? Justifiez.

Correction. Non. Par exemple, toute solution de N_1 est solution de N_2 (Question 1) mais $D_{N_1}^{AC}(X_1) \not\subseteq D_{N_2}^{AC}(X_1)$ (Question 2).

Question 4. Supposons maintenant que toute affectation de X localement cohérente dans N^* est localement cohérente dans N . Est-il forcément vrai que pour tout $X_i \in X$, on a $D_{N^*}^{AC}(X_i) \subseteq D_N^{AC}(X_i)$? Justifiez votre réponse.

Correction. Non. Prenons par exemple X_1, X_2, X_3 avec domaines $\{0, 1\}$ et les réseaux N^*, N obtenus respectivement avec $C^* = \{X_1 \neq X_2, X_2 \neq X_3, X_3 \neq X_1\}$ et $C = \{\text{AllDifferent}(X_1, X_2, X_3)\}$. Pour le réseau N , toutes les affectations de 2 variables ou moins sont localement cohérentes et aucune affectation des 3 variables ne l'est. Comme aucune affectation des 3 variables n'est localement cohérente dans N^* , ces deux réseaux satisfont l'hypothèse de la question. Cependant, appliquer AC sur N va vider tous les domaines alors que N^* est déjà arc cohérent.

Question 5. Revenons aux réseaux N_1 et N_2 des deux premières questions. Est-il possible que $\text{BT}(N_2, \emptyset)$ effectue plus d'appels récursifs que $\text{BT}(N_1, \emptyset)$ en utilisant les mêmes heuristiques de choix de variables et de valeurs ? Justifiez votre réponse.

Correction. Non. Supposons par l'absurde qu'il existe une affectation partielle ϕ de X telle que

- $\text{BT}(N_2, \emptyset)$ appelle $\text{BT}(N_2, \phi)$ à un point de son exécution, et
- $\text{BT}(N_1, \emptyset)$ n'appelle jamais $\text{BT}(N_1, \phi)$ durant son exécution.

Soit ϕ' l'affectation partielle telle que $\text{BT}(N_2, \phi')$ appelle récursivement $\text{BT}(N_2, \phi)$. On a alors $\phi = \phi' \cup \{X_i, v_i\}$ et par minimalité $\text{BT}(N_1, \emptyset)$ appelle $\text{BT}(N_1, \phi')$ à un point de son exécution. Cependant, comme N_1 et N_2 ont le même ensemble de solutions et que toute affectation localement cohérente dans N_2 l'est aussi dans N_1 , $\text{BT}(N_1, \phi' \cup \{X_i, v_i\})$ renverra `false` pour toute valeur v qui précède v_i dans l'heuristique de choix de valeurs. $\text{BT}(N_1, \emptyset)$ appellera donc nécessairement $\text{BT}(N_1, \phi)$ à un point de son exécution, ce qui contredit notre hypothèse initiale.

Exercice 6

Un bloc de la ville de New York a été représenté dans une grille. Chaque case contient un immeuble de 10, 20, 30 ou 40 étages. Les immeubles d'une même rangée, ligne ou colonne, sont tous de tailles différentes. Les nombres donnés sur les bords indiquent le nombre d'immeubles visibles sur la rangée correspondante par un observateur situé à cet endroit. Par exemple, si une ligne contient la disposition 20-40-30-10, deux immeubles sont visibles à partir de la gauche et trois à partir de la droite.

Le jeu consiste à recouper les différentes observations pour retrouver la hauteur de chaque immeuble. La Figure 1 donne un exemple.

Question 1. Exprimez ce jeu sous forme d'un réseau de contraintes. Vous pourrez utiliser des contraintes "de table", c'est-à-dire définies en extension par un ensemble de tuples qu'il faudra lister.

4	1	3	2
2	2	10	40
3	1	20	10
2	2	30	20
1	3	40	30
1	2	10	20
2	2	2	2

Figure 1: A gauche, un exemple du problème ; à droite, une solution.

Correction. Pour cette question, un modèle simple est de placer une variable par immeuble, dont le domaine est l'ensemble des hauteurs possibles $\{10, 20, 30, 40\}$, et une variable par observation dont le domaine contient uniquement la valeur de l'observation en question. Pour chaque ligne/colonne, on ajoute ensuite une contrainte qui force les hauteurs des immeubles à être cohérentes avec les observations.

Formellement, on a 16 variables d'observation $O_1^G, \dots, O_4^G, O_1^D, \dots, O_4^D, O_1^H, \dots, O_4^H, O_1^B, \dots, O_4^B$, 16 variables $X_{i,j}$ pour $i, j \in \{1, \dots, 4\}$, et 8 contraintes

$$\begin{aligned} \forall i \in \{1, \dots, 4\}, \quad & c(O_i^G, X_{i,1}, \dots, X_{i,4}, O_i^D) \\ \forall j \in \{1, \dots, 4\}, \quad & c(O_j^H, X_{1,j}, \dots, X_{4,j}, O_j^B) \end{aligned}$$

où c est la fonction booléenne qui lie les 24 permutations possibles de $(10, 20, 30, 40)$ aux observations associées, dont les tuples sont :

4	10	20	30	40	1	2	30	20	10	40	1
3	10	20	40	30	2	2	30	20	40	10	2
3	10	30	20	40	1	2	30	10	20	40	1
3	10	30	40	20	2	2	30	10	40	20	2
2	10	40	20	30	2	2	30	40	20	10	3
2	10	40	30	20	3	2	30	40	10	20	2
3	20	10	30	40	1	1	40	10	30	20	3
2	20	10	40	30	2	1	40	10	20	30	2
3	20	30	10	40	1	1	40	30	10	20	3
3	20	30	40	10	2	1	40	30	20	10	4
2	20	40	10	30	2	1	40	20	10	30	2
2	20	40	30	10	3	1	40	20	30	10	3

Question 2. (Plus difficile) On considère maintenant le même jeu, mais avec une taille de bloc n arbitraire. Exprimez ce jeu sous la forme d'un réseau de contraintes dont la taille est polynomiale en n et où toutes les contraintes sont d'arité au plus 3.

Pour cette dernière question il n'est pas nécessaire de lister tous les tuples de chaque contrainte ; une description claire et précise est suffisante (par exemple sous la forme d'une formule logique et/ou arithmétique).

Correction. L'approche utilisée pour la question 1 ne fonctionne plus ici car une généralisation naïve du modèle contiendrait des contraintes d'arité $n+2$. On va tout de même réutiliser les variables du premier modèle, avec les mêmes interprétations. Pour exprimer la relation "la valeur de l'observation est le nombre

d'immeubles visibles depuis ce point de vue" avec des contraintes de faible arité, il va être utile d'introduire de nouvelles variables.

Commençons par introduire, pour tout $i, j \in \{1, \dots, n\}$ et $L \in \{G, D, H, B\}$, une variable booléenne $V_{i,j}^L$ qui signifie "l'immeuble $X_{i,j}$ est visible depuis le côté L ", une variable $M_{i,j}^L$ de domaine $\{10, \dots, 10n\}$ qui représente la taille maximum d'un immeuble du côté L de $X_{i,j}$ (inclus), et une variable $K_{i,j}^L$ de domaine $\{1, \dots, n\}$ qui représente le nombre d'immeubles visibles depuis le côté L jusqu'à $X_{i,j}$. On pose alors les contraintes :

$$\begin{aligned} \forall i \in \{1, \dots, n\}, \quad & M_{i,1}^G = X_{i,1} \quad K_{i,1}^G = 1 \\ \forall i \in \{1, \dots, n\}, \quad & M_{i,n}^D = X_{i,n} \quad K_{i,n}^D = 1 \\ \forall j \in \{1, \dots, n\}, \quad & M_{1,j}^H = X_{1,j} \quad K_{1,j}^H = 1 \\ \forall j \in \{1, \dots, n\}, \quad & M_{n,j}^B = X_{n,j} \quad K_{n,j}^B = 1 \\ \forall i, j \in \{1, \dots, n\} \times \{2, \dots, n\}, \quad & M_{i,j}^G = \max(M_{i,j-1}^G, X_{i,j}) \quad K_{i,j}^G = K_{i,j-1}^G + V_{i,j}^G \\ \forall i, j \in \{1, \dots, n\} \times \{1, \dots, n-1\}, \quad & M_{i,j}^D = \max(M_{i,j+1}^D, X_{i,j}) \quad K_{i,j}^D = K_{i,j+1}^D + V_{i,j}^D \\ \forall i, j \in \{2, \dots, n\} \times \{1, \dots, n\}, \quad & M_{i,j}^H = \max(M_{i-1,j}^H, X_{i,j}) \quad K_{i,j}^H = K_{i-1,j}^H + V_{i,j}^H \\ \forall i, j \in \{1, \dots, n-1\} \times \{1, \dots, n\}, \quad & M_{i,j}^B = \max(M_{i+1,j}^B, X_{i,j}) \quad K_{i,j}^B = K_{i+1,j}^B + V_{i,j}^B \end{aligned}$$

Il reste alors à exprimer le fait que tous les immeubles sont de hauteurs différentes sur une même rangée, et de s'assurer que V_{ij}^L vaut 1 si et seulement si X_{ij} est visible depuis le côté L . Pour cela, on pose les contraintes :

$$\begin{aligned} \forall i, j, k \in \{1, \dots, n\}^3 \text{ tels que } j \neq k, \quad & X_{i,j} \neq X_{i,k} \\ \forall i, j, k \in \{1, \dots, n\}^3 \text{ tels que } i \neq k, \quad & X_{i,j} \neq X_{k,j} \\ \forall i, j \in \{1, \dots, n\}^2, \quad & V_{i,j}^G \iff (X_{i,j} = M_{i,j}^G) \\ \forall i, j \in \{1, \dots, n\}^2, \quad & V_{i,j}^D \iff (X_{i,j} = M_{i,j}^D) \\ \forall i, j \in \{1, \dots, n\}^2, \quad & V_{i,j}^H \iff (X_{i,j} = M_{i,j}^H) \\ \forall i, j \in \{1, \dots, n\}^2, \quad & V_{i,j}^B \iff (X_{i,j} = M_{i,j}^B) \end{aligned}$$

et finalement, on s'assure que les observations correspondent aux nombres d'immeubles visibles sur chaque rangée :

$$\begin{aligned} \forall i \in \{1, \dots, n\}, \quad & K_{i,n}^G = O_i^G \quad K_{i,1}^D = O_i^D \\ \forall j \in \{1, \dots, n\}, \quad & K_{n,j}^H = O_j^H \quad K_{1,j}^B = O_j^B \end{aligned}$$

Université de Montpellier - Master 2

Module **Contraintes**

Feuille TD 2 - 23/10/2023

Exercice 1

Soit la contrainte globale $\text{ValueCount}([X_1, \dots, X_n], [Y_1, \dots, Y_q])$ qui est satisfaite si et seulement si pour chaque valeur $v \in \{1, \dots, q\}$ on a $Y_v = |\{i \in \{1, \dots, n\} \mid X_i = v\}|$. Par exemple, $([2, 1, 5, 1, 2, 3, 1], [3, 2, 1])$ satisfait la contrainte ValueCount alors que $([1, 1, 2, 3, 2], [2, 3, 1])$ ne la satisfait pas.

Question 1. Soit une contrainte globale $\text{AllDifferent}(X_1, \dots, X_n)$ telle que l'union des domaines forme un intervalle $[1, \dots, m]$. Montrez que l'on peut exprimer cette contrainte à l'aide d'une contrainte ValueCount portant sur un nombre de variables polynomial en $n + m$.

Correction. La contrainte $\text{AllDifferent}(X_1, \dots, X_n)$ peut s'exprimer de manière équivalente par la condition “toute valeur $v \in \cup_{i \leq n} D(X_i)$ apparaît au plus une fois dans le vecteur (X_1, \dots, X_n) ”. En utilisant la contrainte ValueCount et le fait que l'union des domaines forme un intervalle $[1, \dots, m]$, on peut donc reformuler $\text{AllDifferent}(X_1, \dots, X_n)$ par

$$\text{ValueCount}([X_1, \dots, X_n], [Y_1, \dots, Y_m])$$

avec pour tout $i \leq m$, $D(Y_i) = \{0, 1\}$.

Question 2. Justifiez brièvement que si la contrainte globale AllDifferent admet une AC-décomposition lorsque l'union des domaines des variables forme un intervalle $[1, \dots, m]$, alors AllDifferent admet une AC-décomposition dans le cas général.

Correction. Soit $\text{AllDifferent}(X_1, \dots, X_n)$ une contrainte (sans hypothèse sur les domaines), m le nombre de valeurs distinctes qui apparaissent dans les domaines de X_1, \dots, X_n et ϕ une bijection quelconque de $\cup_{i \leq n} D(X_i)$ dans $\{1, \dots, m\}$. On a alors

$$\text{AllDifferent}(X_1, \dots, X_n) \iff \text{AllDifferent}(\phi(X_1), \dots, \phi(X_n))$$

où $\phi(X_i)$ désigne la variable obtenue à partir de X_i en remplaçant chaque $v \in D(X_i)$ par $\phi(v)$. Toute AC-décomposition de $\text{AllDifferent}(\phi(X_1), \dots, \phi(X_n))$ (dont l'union des domaines forme par construction un intervalle) peut donc être transformée en AC-décomposition de $\text{AllDifferent}(X_1, \dots, X_n)$ en remplaçant chaque valeur $v \in \{1, \dots, m\}$ par $\phi^{-1}(v)$.

Question 3. Calculer l'arc cohérence sur la contrainte globale ValueCount est NP-difficile. Il n'existe donc pas de AC-décomposition pour ValueCount si $P \neq NP$. S'il l'on suppose au contraire que $P = NP$, peut-on espérer que ValueCount admette une AC-décomposition ? Justifiez votre réponse.

Correction. En combinant les réponses aux questions précédentes :

ValueCount admet une AC-décomposition

- ⇒ AllDifferent admet une AC-décomposition lorsque l'union des domaines forme un intervalle (Q1)
- ⇒ AllDifferent admet une AC-décomposition dans le cas général (Q2)

ce qui est faux d'après le cours (indépendamment des classes P et NP). ValueCount n'admet donc pas d'AC-décomposition, même si $P = NP$.

Question 4. On a vu en cours que calculer l'arc cohérence sur la contrainte globale $\text{NValue}(X_1, \dots, X_n, N)$ (qui est satisfaite si et seulement si $N = |\{X_1, \dots, X_n\}|$) est NP-difficile. Supposons à nouveau que $P = NP$. Peut-on espérer que NValue admette une AC-décomposition ? Justifiez votre réponse.

Correction. Non car on peut exprimer AllDifferent avec une contrainte NValue :

$$\text{AllDifferent}(X_1, \dots, X_n) \iff \text{NValue}(X_1, \dots, X_n, N)$$

où la variable N a pour domaine $\{n\}$. AllDifferent n'est pas AC-décomposable, donc NValue ne l'est pas non plus, même si $P = NP$.

Question 5. Soit la contrainte globale $\text{MultisetEqual}([X_1, \dots, X_n], [Y_1, \dots, Y_n])$ qui est satisfaite si et seulement si les multisets $\{\{X_i \mid i \in \{1, \dots, n\}\}\}$ et $\{\{Y_i \mid i \in \{1, \dots, n\}\}\}$ sont égaux. (Un multiset diffère d'un ensemble par le fait qu'il tient compte du nombre d'occurrences de chaque élément.) Par exemple $([1, 1, 1, 2, 3], [2, 1, 3, 1, 1])$ satisfait la contrainte MultisetEqual alors que $([1, 1, 1, 2, 3], [2, 2, 3, 1, 1])$ ne la satisfait pas. Contrairement à ValueCount et NValue , on sait que calculer l'arc cohérence est polynomial sur MultisetEqual . MultisetEqual admet-elle une AC-décomposition ? Justifiez votre réponse.

Correction. Si l'on considère n variables X_1, \dots, X_n telles que l'union des domaines forme un intervalle $\{1, \dots, m\}$ avec $k = m - n \geq 0$, alors on a

$$\text{AllDifferent}(X_1, \dots, X_n) \iff \text{MultisetEqual}([X_1, \dots, X_n, Z_1, \dots, Z_k], [Y_1, \dots, Y_m])$$

où le domaine de chaque Z_i est $\{1, \dots, m\}$ et le domaine de chaque Y_i est $\{i\}$. Par la réponse à la question 2, AllDifferent n'est pas AC-décomposable même lorsque l'union des domaines forme un intervalle, donc MultisetEqual ne l'est pas non plus.

Exercice 2

Soit le réseau de contraintes $N = (X, D, C)$, où $X = \{X_1, \dots, X_5\}$, $D(X_1) = D(X_2) = \{0, 2, 4, 6\}$, $D(X_3) = \{2, 5, 6, 7\}$, $D(X_4) = \{0, 2, 6, 8, 12\}$, $D(X_5) = \{0, 1, 6\}$, $c_1 \equiv X_1 + X_2 + X_3 = X_4$, $c_2 \equiv |X_1 - X_3| = X_5$, $c_3 \equiv |X_1 - X_2| = 4$ et $C = \{c_1, c_2, c_3\}$.

Question 1. Appliquez BC sur N.

Correction. On examine les contraintes une par une, en supprimant les bornes des domaines qui n'ont pas de support BC. Attention, un support BC peut affecter certaines variables à des valeurs qui ne sont pas dans leur domaine ; il faut juste que la valeur se trouve entre les bornes !

Notation pour cet exercice : “ $c_i : (X_j, v)$ ” signifie “on supprime la valeur $v \in D(X_j)$ car elle n'a pas de support BC dans la contrainte c_i ”.

$$c_1 : (X_4, 0)$$

Toutes les bornes restantes ont un support BC pour chaque contrainte. La fermeture BC de D est donc :

$$\begin{aligned} D(X_1) &= D(X_2) = \{0, 2, 4, 6\} \\ D(X_3) &= \{2, 5, 6, 7\} \\ D(X_4) &= \{2, 6, 8, 12\} \\ D(X_5) &= \{0, 1, 6\} \end{aligned}$$

Question 2. Appliquez AC sur N.

Correction. Avec les mêmes notations que dans le TD1 :

$$\begin{aligned} c_1 &: (X_3, 5), (X_3, 7), (X_4, 0) \\ c_2 &: (X_5, 1), (X_1, 4) \\ c_3 &: (X_2, 0) \\ c_4 &: (X_4, 2) \end{aligned}$$

Toutes les valeurs restantes ont un support pour chaque contrainte. La fermeture AC de D est donc :

$$\begin{aligned} D(X_1) &= \{0, 2, 6\} \\ D(X_2) &= \{2, 4, 6\} \\ D(X_3) &= \{2, 6\} \\ D(X_4) &= \{6, 8, 12\} \\ D(X_5) &= \{0, 6\} \end{aligned}$$

Question 3. Appliquez SAC sur N.

Correction. On part des domaines AC de la question 2 car toute valeur supprimée par AC l'est aussi par SAC. On fixe $D(X_1) = \{0\}$ et on applique AC :

$$\begin{aligned} &[\text{Sous l'hypothèse } D(X_1) = \{0\}] c_2 : (X_3, 2), (X_5, 0) \\ &[\text{Sous l'hypothèse } D(X_1) = \{0\}] c_3 : (X_2, 2), (X_2, 6) \\ &[\text{Sous l'hypothèse } D(X_1) = \{0\}] c_4 : \text{domaines vides pour } X_1, X_2, X_3, X_4 \end{aligned}$$

On a au moins un domaine vide, donc SAC supprime 0 du domaine de X_1 . On recommence en fixant cette fois-ci $D(X_1) = \{2\}$:

$$\begin{aligned} &[\text{Sous l'hypothèse } D(X_1) = \{2\}] c_3 : (X_2, 2), (X_2, 4) \\ &[\text{Sous l'hypothèse } D(X_1) = \{2\}] c_2 : (X_3, 6), (X_5, 6) \\ &[\text{Sous l'hypothèse } D(X_1) = \{2\}] c_4 : \text{domaines vides pour } X_1, X_2, X_3, X_4 \end{aligned}$$

SAC supprime donc également 2 du domaine de X_1 , qui devient alors le singleton $\{6\}$. Par AC on a finalement

$$\begin{aligned} c_3 &: (X_2, 4), (X_2, 6) \\ c_2 &: (X_3, 2), (X_5, 6) \\ c_4 &: \text{domaines vides pour } X_1, X_2, X_3, X_4 \end{aligned}$$

et après SAC tous les domaines sont donc vides : $\forall i \leq 5, D(X_i) = \emptyset$.

Soit N^* le réseau de contraintes obtenu à partir de N en fixant les domaines à $D(X_1) = D(X_2) = \{0, \dots, 6\}$, $D(X_3) = \{2, \dots, 7\}$, $D(X_4) = \{0, \dots, 12\}$, $D(X_5) = \{0, \dots, 6\}$.

Question 4. Appliquez AC et BC sur N^* . Comparez AC et BC sur les réseaux de contraintes dont les domaines sont des intervalles de \mathbb{Z} .

Correction. La fermeture BC de N^* est

$$\begin{aligned} D(X_1) &= \{0, \dots, 6\} \\ D(X_2) &= \{0, \dots, 6\} \\ D(X_3) &= \{2, \dots, 7\} \\ D(X_4) &= \{2, \dots, 12\} \\ D(X_5) &= \{0, \dots, 6\} \end{aligned}$$

et la fermeture AC de N^* est presque identique, à l'exception près que la valeur 3 est enlevée des domaines de X_1 et X_2 à cause de la contrainte c_3 . AC reste donc une propriété strictement plus forte que BC, même lorsque les domaines sont des intervalles de \mathbb{Z} .

Exercice 3

Pour $q \geq k \geq 1$, on considère la contrainte globale (k, q) -ConsecutiveOnes($[X_1, \dots, X_n]$) qui porte sur n variables booléennes X_1, \dots, X_n . Cette contrainte est satisfaite si et seulement si chaque séquence maximale (non vide) de 1 consécutifs dans le vecteur (X_1, \dots, X_n) est de longueur au moins k et au plus q . Par exemple, $([0, 1, 1, 0, 1, 1, 1])$ satisfait $(2, 3)$ -ConsecutiveOnes mais $([0, 1, 0, 0, 1, 1, 1])$ ne la satisfait pas.

Question 1. Montrez que pour tout $q \geq k \geq 1$, la contrainte globale (k, q) -ConsecutiveOnes admet une AC-décomposition.

Correction. On va construire une AC-décomposition de (k, q) -ConsecutiveOnes(X_1, \dots, X_n) en s'inspirant de celle de AtLeast- p -v vue en cours. On ajoute n variables C_1, \dots, C_n de domaine $\{0, \dots, q\}$, dont le but va être de compter le nombre de 1 consécutifs dans les séquences de X_1, \dots, X_n . On fixe $X_1 = C_1$, et pour chaque $i \in \{2, \dots, n\}$ on impose une contrainte :

$$c_{(i-1, i)} : ((X_i = 1) \wedge (C_i = C_{i-1} + 1)) \vee ((X_i = 0) \wedge (C_i = 0) \wedge (C_{i-1} \in \{0, k, \dots, q\}))$$

Le sens de cette contrainte est assez intuitif. Si $X_i = 1$, alors le compteur pour la séquence de 1 courante est incrémenté. Si $X_i = 0$, le compteur est fixé à zéro et on s'assure que C_{i-1} (qui contient alors le nombre de 1 consécutifs dans la séquence maximale de 1 qui se termine en X_{i-1}) est soit 0 soit dans l'intervalle $[k, \dots, q]$. Il reste alors à vérifier la séquence qui se termine en X_n , et pour cela on fixe $D(C_n) = \{0, k, \dots, q\}$.

La conjonction de ces contraintes est équivalente à (k, q) -ConsecutiveOnes(X_1, \dots, X_n), le réseau est de taille polynomiale, l'arité est bornée par 3 et le graphe d'incidence est Berge-acyclique : c'est donc une AC-décomposition.

La contrainte globale (k, q) -ConsecutiveValues($[X_1, \dots, X_n]$) porte sur n variables entières et est satisfaite si et seulement si chaque séquence maximale (non vide) de valeurs identiques consécutives est de longueur au moins k et au plus q .

Question 2. La contrainte globale (k, q) -ConsecutiveValues admet-elle une AC-décomposition ? Justifiez votre réponse.

Correction. Oui. On construit une AC-décomposition de (k, q) -ConsecutiveValues(X_1, \dots, X_n) dans le même esprit que pour la question précédente. Soit S l'union des domaines de X_1, \dots, X_n . On ajoute n variables C_1, \dots, C_n de domaine $S \times \{1, \dots, q\}$, qui vont garder en mémoire la dernière valeur observée et la longueur de la séquence courante. On remarquera que chaque valeur du domaine de C_1 est une paire (v, l) . Pour chaque $i \in \{2, \dots, n\}$ on impose une contrainte $c(C_{i-1}, C_i, X_i)$ définie comme suit :

$$((v_1, l_1), (v_2, l_2), v) \in c \iff \begin{cases} (v_1 = v_2 = v) \wedge (l_2 = l_1 + 1), \text{ ou} \\ (v_1 \neq v_2 = v) \wedge (l_2 = 1) \wedge (l_1 \in \{k, \dots, q\}) \end{cases}$$

Tout d'abord, on observe que $X_i = v$ implique que C_i doit être affecté à une paire ayant v pour premier élément (réflété par le fait que $v_2 = v$ dans tous les tuples de c). Les tuples de $c(C_{i-1}, C_i, X_i)$ sont divisés en deux groupes. Dans le premier, X_i est égal à X_{i-1} et le compteur l est incrémenté. Dans le second, X_i est différent de X_{i-1} et le compteur est remis à 1; cela arrive à la fin de chaque séquence (sauf la dernière) et donc on vérifie que sa longueur est bien dans l'intervalle $\{k, \dots, q\}$.

Il reste à ajouter des contraintes pour initialiser C_1 et vérifier la longueur de la dernière séquence. On impose donc $C_1 = (X_1, 1)$ et $D(C_n) = S \times \{k, \dots, q\}$. Pour les mêmes raisons que la question précédente, ce réseau de contraintes est une AC-décomposition de (k, q) -ConsecutiveValues(X_1, \dots, X_n).

La contrainte globale DifferentConsecutiveLengths($[X_1, \dots, X_n]$) porte sur n variables entières et est satisfaite si et seulement si pour toute valeur v , il n'existe pas deux séquences maximales (non vides) de v consécutifs qui ont la même longueur.

Question 3. La contrainte globale DifferentConsecutiveLengths admet-elle une AC-décomposition ? Justifiez votre réponse.

Correction. Non car on peut exprimer la contrainte AllDifferent(X_1, \dots, X_n) à partir de cette contrainte :

$$\text{AllDifferent}(X_1, \dots, X_n) \iff \text{DifferentConsecutiveLengths}(X_1, Z_1, X_2, Z_2, \dots, Z_{n-1}, X_n)$$

où le domaine de chaque variable Z_i est composé d'une valeur quelconque qui n'apparaît dans aucun autre domaine. AllDifferent n'est pas AC-décomposable, donc DifferentConsecutiveLengths ne l'est pas non plus.

Question 4. Les réponses aux questions 2 et 3 vous permettent-elles de déterminer si l'arc cohérence sur ces deux contraintes est calculable en temps polynomial ?

Correction. Par la question 2, on peut déduire que l'AC sur (k, q) -ConsecutiveValues est calculable en temps polynomial. Pour DifferentConsecutiveLengths, la réponse à la question 3 ne permet pas de conclure car il existe des contraintes pour lesquelles on peut calculer l'AC en temps polynomial mais qui ne sont pas AC-décomposables (e.g. AllDifferent).

Exercice 4

Dans un réseau de contraintes $N = (X, D, C)$, on dit qu'une valeur $v \in D(X_i)$ est singleton viable si le réseau de contraintes obtenu à partir de N' en fixant $D(X_i) = \{v\}$ a une fermeture arc cohérente non vide. Un réseau est singleton arc cohérent (SAC) si toutes les valeurs sont singleton viables; "appliquer SAC" désigne le processus d'enlever des valeurs qui ne sont pas singleton viables jusqu'à ce que toutes le soient.

Soit le réseau de contraintes $N^* = (X, D, C)$, où $X = \{X_1, \dots, X_4\}$, $D(X_1) = D(X_2) = \{1, 2, 7, 8\}$, $D(X_3) = D(X_4) = \{1, 2, 3, 4\}$, $c_1 \equiv |X_1 - X_2| = |X_3 - X_4|$, $c_2 \equiv X_3 = 2X_2$, $c_3 \equiv X_1 + X_4 = 5$ et $C = \{c_1, c_2, c_3\}$.

Question 1. Appliquez BC sur N^* .

Correction.

$$\begin{aligned} c_2 : & (X_3, 1), (X_2, 7), (X_2, 8) \\ c_3 : & (X_1, 7), (X_1, 8), (X_4, 1), (X_4, 2) \end{aligned}$$

Toutes les bornes restantes ont un support BC pour chaque contrainte. La fermeture BC de D est donc :

$$\begin{aligned} D(X_1) &= D(X_2) = \{1, 2\} \\ D(X_3) &= \{2, 3, 4\} \\ D(X_4) &= \{3, 4\} \end{aligned}$$

Question 2. Appliquez SAC sur N^* .

Correction. On part des domaines BC de la question 1. La valeur 3 ∈ $D(X_3)$ n'est pas viable pour la contrainte c_2 , donc elle n'est pas singleton viable non plus. On fixe $D(X_1) = \{1\}$ et on applique AC :

$$\begin{aligned} & [\text{Sous l'hypothèse } D(X_1) = \{1\}] c_3 : (X_4, 3) \\ & [\text{Sous l'hypothèse } D(X_1) = \{1\}] c_1 : (X_3, 2), (X_2, 2) \\ & [\text{Sous l'hypothèse } D(X_1) = \{1\}] c_2 : (X_2, 1), (X_3, 4) \end{aligned}$$

La dernière étape dérive des domaines vides pour X_3 et X_2 , donc SAC supprime la valeur 1 du domaine de X_1 . Par AC on a ensuite :

$$\begin{aligned} c_3 : & (X_4, 4) \\ c_1 : & (X_2, 2), (X_2, 2) \\ c_2 : & (X_3, 4) \end{aligned}$$

Les domaines sont alors

$$\begin{aligned} D(X_1) &= \{2\} \\ D(X_2) &= \{1\} \\ D(X_3) &= \{2\} \\ D(X_4) &= \{3\} \end{aligned}$$

qui est singleton arc cohérent.

Question 3. Combien comporte de variables le plus petit réseau de contraintes binaires normalisé (c-à-d tel que deux contraintes ne peuvent pas avoir la même portée) qui est SAC mais n'a pas de solution ? Justifiez.

Correction. Le plus petit réseau de contraintes binaires normalisé qui est SAC mais n'a pas de solution a 4 variables. En effet, pour trois variables ou moins toute valeur singleton viable s'étend à une solution. En revanche, il est facile de vérifier que la clique de différences sur quatre variables

$$\begin{cases} X_1 \neq X_2 \\ X_1 \neq X_3 \\ X_1 \neq X_4 \\ X_2 \neq X_3 \\ X_2 \neq X_4 \\ X_3 \neq X_4 \end{cases}$$

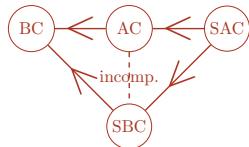
avec un domaine $\{1, 2, 3\}$ pour toutes les variables est SAC mais n'a pas de solution.

Question 4. On peut définir la propriété SBC de façon analogue à SAC, en remplaçant AC par BC dans la définition. Comparez les propriétés AC, SAC, BC et SBC.

Correction. Par le cours, SAC est strictement plus fort que AC, qui est strictement plus fort que BC. Maintenant, comparons AC et SBC. AC n'est pas plus fort que SBC car le réseau $(X_1 = X_2), (X_2 \neq X_1)$ avec des domaines $\{0, 1\}$ est AC mais pas SBC. De même, SBC n'est pas plus fort que AC car le réseau composé d'une unique contrainte $2 \in \{X_1, X_2, X_3\}$ avec des domaines $\{1, 3\}$ est SBC mais pas AC. SAC et AC sont donc incomparables.

Comme SBC est au moins aussi fort que BC et incomparable avec AC (qui est strictement plus fort que BC), SBC est strictement plus fort que BC. De même, SAC est au moins aussi fort que SBC et SBC est incomparable avec AC (qui est strictement plus faible que SAC), donc SAC est strictement plus fort que SBC.

Visuellement, on a donc :



Un réseau de contraintes binaires $N = (X, D, C)$ est k -quasi-arborescent s'il est normalisé et qu'il existe k variables dont la suppression (ainsi que la suppression des contraintes incidentes) produit un réseau arborescent. On rappelle qu'un réseau arborescent AC admet toujours une solution.

Question 5. Un réseau 1-quasi-arborescent qui est SAC admet-il toujours une solution ? Justifiez.

Correction. Oui. En effet, soit N un réseau 1-quasi-arborescent et SAC, X_1 une variable de N dont la suppression produit un réseau arborescent et $v \in D(X_1)$ une valeur singleton viable dans son domaine. Par définition, le réseau N' obtenu en fixant $D(X_1) = v$ est arc cohérent. De plus, les contraintes comprenant X_1 dans leur portée peuvent être supprimées sans conséquences car elles sont satisfaites par toute affectation des variables à des valeurs de leur domaine. Le réseau ainsi obtenu est arborescent et arc-cohérent, donc il a une solution (voir TD 1) qui est également une solution de N .

Question 6. Soit $k \geq 1$. Peut-on résoudre en temps polynomial les réseaux de contraintes k -quasi-arborescents ? Justifiez.

Correction. Oui. En effet, soit N un réseau k -quasi-arborescent. On commence par énumérer les sous-ensembles de variables de taille k jusqu'à en trouver un dont la suppression produit un réseau arborescent. On note S ce sous-ensemble de variables.

Maintenant, on énumère toutes les affectations possibles de S . Pour chaque affectation ϕ , on fixe chaque variable $X \in S$ à la valeur $\phi(X)$ et on applique l'arc cohérence. Puisque le réseau résiduel (après affectation de S) est arborescent, si l'arc cohérence ne vide pas les domaines pour au moins une affectation ϕ alors on peut déduire que N admet une solution. Réciproquement, si l'arc cohérence dérive au moins un domaine vide pour toute affectation ϕ alors aucune affectation de S ne peut s'étendre à une solution de N , et donc N n'admet pas de solution.

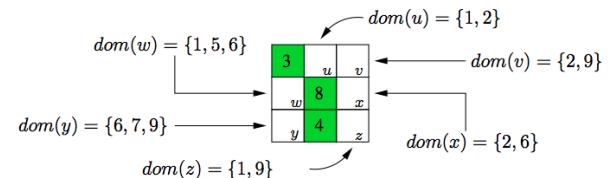
Il y a $O(n^k)$ sous-ensembles de variables S à énumérer, où n est le nombre de variables. Pour chacun, on peut vérifier en temps linéaire si le réseau résiduel (après suppression) est arborescent et il y a $O(d^k)$

affectations à énumérer, où d est la taille maximum des domaines. L'algorithme ci-dessus est donc polynomial pour un k fixé.

Exercice 5

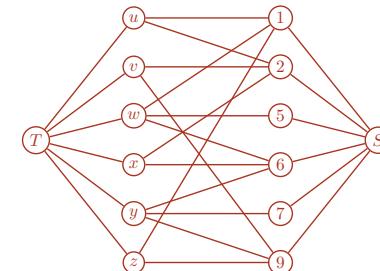
Soit le réseau de contraintes $N = (X, D, C)$, où $X = \{u, v, w, x, y, z\}$, $C = \{\text{AllDifferent}(u, v, w, x, y, z)\}$ et les domaines sont définis comme ci-dessous.

Question 1. Donner l'état des domaines après fermeture GAC pour les domaines suivants :

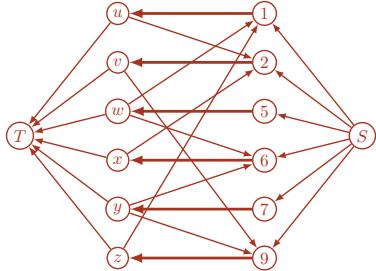


Correction. Les questions de cet exercice peuvent être traitées de deux façons différentes : soit en revenant à la définition de l'arc cohérence et en cherchant des supports pour chaque valeur, soit en appliquant l'algorithme d'arc cohérence spécialisé pour la contrainte `AllDifferent` entrevu en cours. La deuxième approche est plus instructive (et vous devriez être déjà familiers avec la première), donc c'est celle que nous verrons dans cette correction.

On commence par tracer le graphe des valeurs de la contrainte :



Il faut ensuite calculer un flot maximum de S vers T . Cela revient à calculer un couplage maximum dans le graphe biparti ayant les variables à gauche et les valeurs à droite. Ici, le couplage qui associe chaque variable à la valeur qui lui fait directement face est parfait. Ce couplage est de taille $|X|$, donc la contrainte admet au moins une affectation cohérente (donnée par le couplage). On oriente les arêtes utilisées par le flot de droite à gauche, et toutes les autres arêtes dans l'autre sens (le couplage utilisé est dessiné en surbrillance) :

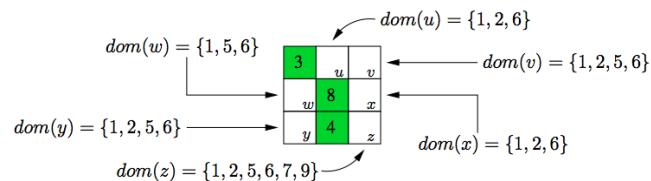


On identifie maintenant les composantes fortement connexes du graphe. Ici, on en a sept : $\{u, v, z, 1, 2, 9\}$, $\{y\}$, $\{7\}$, $\{5\}$, $\{w\}$, $\{6\}$ et $\{x\}$. Finalement, on supprime des domaines toutes les valeurs correspondant à des arêtes connectant des composantes connexes distinctes, à l'exception des arêtes du couplage. Les domaines résiduels sont alors :

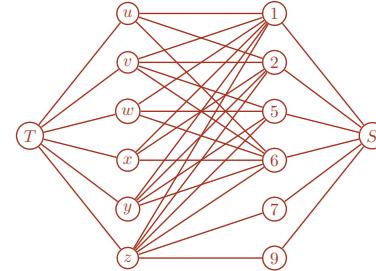
$$\begin{aligned} D(u) &= \{1, 2\} \\ D(v) &= \{2, 9\} \\ D(w) &= \{5\} \\ D(x) &= \{6\} \\ D(y) &= \{7\} \\ D(z) &= \{1, 9\} \end{aligned}$$

ce qui est la fermeture AC de N .

Question 2. Donner l'état des domaines après fermeture GAC pour les domaines suivants :



Correction. On trace le graphe des valeurs :



Cette fois-ci, il n'existe pas de couplage de taille $|X|$ car $|X| = |D|$ et les valeurs 7 et 9 ont z comme unique variable voisine. Comme les couplages de taille $|X|$ du graphe des valeurs sont en correspondance avec les affectations de X qui satisfont la contrainte AllDiff, cette contrainte n'a aucun support pour aucune valeur ; la fermeture AC de N est donc un ensemble de domaines vides.

Université de Montpellier - Master 2
Module **Contraintes**

Feuille TD 3 - 13/11/2023

Exercice 1

Soit $\Delta = \{c_0, c_1, c_2, c_3, \mu_0, \mu_1\}$ un langage de contraintes sur le domaine $D = \{0, 1\}$ dont les fonctions sont définies comme suit :

$$\begin{aligned} c_0(x, y, z) &= x \vee y \vee z = D^3 \setminus \{(0, 0, 0)\} \\ c_1(x, y, z) &= \bar{x} \vee y \vee z = D^3 \setminus \{(1, 0, 0)\} \\ c_2(x, y, z) &= \bar{x} \vee \bar{y} \vee z = D^3 \setminus \{(1, 1, 0)\} \\ c_3(x, y, z) &= \bar{x} \vee \bar{y} \vee \bar{z} = D^3 \setminus \{(1, 1, 1)\} \\ \mu_0(x) &= \{(0)\} \\ \mu_1(x) &= \{(1)\} \end{aligned}$$

Question 1. Démontrez que $\text{CSP}(\Delta)$ est NP-complet.

On suppose que $P \neq NP$. On dit qu'un langage $\Gamma_1 \subseteq \Delta$ est Δ -maximal si $\text{CSP}(\Gamma_1)$ est polynomial mais que $\text{CSP}(\Gamma_2)$ est NP-complet pour tout Γ_2 tel que $\Gamma_1 \subset \Gamma_2 \subseteq \Delta$.

Question 2. Justifiez qu'il existe au plus six langages Δ -maximaux.

Question 3. Déterminez tous les langages Δ -maximaux.

Exercice 2

Pour tout entier naturel $n > 0$ on définit $X_n = \{x_i \mid 1 \leq i \leq n\}$, $Y_n = \{y_i \mid 1 \leq i \leq n\}$, $\mathcal{Y}_n^{+1} = \{Y_n \cup \{x\} \mid x \in X_n\}$ et $\mathcal{X}_n^{+1} = \{X_n \cup \{y\} \mid y \in Y_n\}$. On considère la famille \mathcal{H} des hypergraphes dont l'ensemble des sommets est de la forme $X_n \cup Y_n$ (pour un n qui peut varier d'un hypergraphe à un autre) et dont les arêtes sont des éléments de $\mathcal{Y}_n^{+1} \cup \mathcal{X}_n^{+1}$.

Question 1. La treewidth des hypergraphes de \mathcal{H} est-elle bornée par une constante ? Justifiez.

Question 2. Démontrez que tout $H \in \mathcal{H}$ a une hypertreewidth d'au plus 2.

Question 3. Soient $n > 0$ et H_n l'hypergraphe dont les sommets sont $X_n \cup Y_n$ et les arêtes sont $\mathcal{Y}_n^{+1} \cup \mathcal{X}_n^{+1}$. Montrer que l'hypertreewidth fractionnaire de H_n est strictement inférieure à 2.

Question 4. La famille \mathcal{H} a la propriété d'être *hypertreewidth-monotone*, c'est-à-dire que retirer une arête d'un hypergraphe de H ne peut pas augmenter son hypertreewidth. Est-ce vrai en général, pour tous les

hypergraphes ? Est-ce vrai en général pour la treewidth ?

Question 5. Dans l'autre sens, il est évidemment possible qu'ajouter une arête à un hypergraphe augmente son hypertreewidth. Montrez que dans ce cas, son hypertreewidth augmente de 1 au maximum.

Exercice 3

Pour tout entier naturel $k > 0$ et tout ensemble $S \subseteq \{0, \dots, k\}$, on définit la fonction booléenne $c_S^k = \{\tau \in \{0, 1\}^k \mid \sum_{i=1}^k \tau[i] \in S\}$. Par exemple, on a

$$c_{\{1, 3\}}^3(x, y, z) = \begin{bmatrix} x & y & z \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad c_{\{0, 1\}}^3(x, y, z) = \begin{bmatrix} x & y & z \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad c_{\{1\}}^2(x, y) = \begin{bmatrix} x & y \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Question 1. Soient $k > 0$ et $S \subseteq \{0, \dots, k\}$ tel que $S \neq \emptyset$ et $0 \notin S$. Démontrez que l'opération \vee ("ou" logique) est un polymorphisme de c_S^k si et seulement si il existe $j \in \{1, \dots, k\}$ tel que $S = \{j, \dots, k\}$.

Question 2. Soient $k > 0$ et $S \subseteq \{0, \dots, k\}$. Montrez que l'opération \vee est un polymorphisme de c_S^k si et seulement si c'est un polymorphisme de $c_{S \setminus \{0\}}^k$. Déduisez-en une caractérisation précise des fonctions c_S^k qui admettent le polymorphisme \vee , en fonction de l'ensemble S .

Question 3. Soit $k > 0$. Pour tout ensemble $S \subseteq \{0, \dots, k\}$, on définit $S^\perp = \{k - i \mid i \in S\}$. Montrez que \vee est un polymorphisme de c_S^k si et seulement si \wedge est un polymorphisme de $c_{S^\perp}^k$.

On rappelle que $\text{CSP}(\Gamma)$ est résolu par AC si la fermeture arc cohérente de toute instance insatisfiable de $\text{CSP}(\Gamma)$ contient un domaine vide.

Question 4. Soit Γ un langage booléen. Démontrez que $\text{CSP}(\Gamma)$ est résolu par AC si et seulement si Γ admet un polymorphisme parmi $\{0, 1, \wedge, \vee\}$.

Question 5. Déduisez-en une caractérisation des langages finis $\Gamma \subset \{c_S^k \mid k > 0, S \subseteq \{0, \dots, k\}\}$ résolus par AC.

Exercice 4

On dit qu'une fonction booléenne $c : D^2 \rightarrow \{0, 1\}$ est ZOA (Zero-One-All) s'il existe $D_1, D_2 \subseteq D$ tels que

- (1) : $\forall (d_1, d_2) \in c$, on a $d_1 \in D_1$ et $d_2 \in D_2$
- (2) : $\forall \alpha \in D_1, |\{d \in D_2 \mid (\alpha, d) \in c\}| \in \{1, |D_2|\}$
- (3) : $\forall \beta \in D_2, |\{d \in D_1 \mid (d, \beta) \in c\}| \in \{1, |D_1|\}$

Par exemple, considérons les fonctions

$$c_1(x, y) = \begin{bmatrix} x & y \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 2 & 1 \end{bmatrix} \quad c_2(x, y) = \begin{bmatrix} x & y \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 2 & 1 \\ 3 & 3 \end{bmatrix}$$

définies sur $D = \{1, 2, 3\}$. Informellement, la propriété ZOA signifie que chaque valeur qui apparaît dans une colonne est compatible avec soit *toutes* les valeurs apparaissant dans l'autre colonne, soit *une seule*. Ainsi, la fonction c_1 est ZOA avec $D_1 = \{1, 2\}$ et $D_2 = \{1, 2, 3\}$, mais la fonction c_2 ne l'est pas car on a $(1, 3), (3, 3) \in c_2$ mais $(2, 3) \notin c_2$.

Question 1. Soit D un domaine fini et l'opération $f_{dd} : D^3 \rightarrow D$ définie par

$$f_{dd}(x, y, z) = \begin{cases} x & \text{si } y \neq z \\ z & \text{sinon.} \end{cases}$$

Démontrez que si une fonction booléenne $c : D^2 \rightarrow \{0, 1\}$ est ZOA, alors f_{dd} est un polymorphisme de c .

Question 2. Soit Γ un langage dont toutes les fonctions sont ZOA. En vous appuyant sur la réponse à la question précédente, démontrez que Γ a la propriété de *bounded width*. Quel algorithme utiliserez-vous pour résoudre $\text{CSP}(\Gamma)$?

Exercice 5

Pour tout entier naturel $n \geq 2$ on définit l'hypergraphe H_n dont l'ensemble des sommets est $\{x_1, \dots, x_n, y_1, \dots, y_n\}$ et l'ensemble des arêtes est $\{\{x_i, x_{i+1}, y_{i+1}\} \mid 1 \leq i \leq n-1\} \cup \{\{x_i, y_i, y_{i+1}\} \mid 1 \leq i \leq n-1\}$.

Question 1. Démontrez que pour tout $n \geq 2$, l'hypertreewidth de H_n est égale à 1.

Question 2. Existe-t-il un entier $n \geq 2$ et un réseau de contraintes N dont l'hypergraphe est H_n , tels que N n'a pas de solution mais appliquer la 3-cohérence forte sur N ne vide aucun domaine ? Justifiez votre réponse.

Qualitative Constraint Network: Basic Notions

Travaux dirigés du cours *Contraintes* (HAI910I)

Michael Sioutis

20 Nov 2023

1 Reminder of Basic Notions

Spatial or temporal information for a set of entities can be represented by a qualitative constraint network (QCN).

Definition 1. A qualitative constraint network (QCN) of some qualitative constraint language is a tuple (V, C) where:

- V is a set of variables over the infinite domain D of the language;
- and C is a mapping $C : V \times V \rightarrow 2^B$ associating a relation (set of base relations) of the language with each pair of variables in V .

Typically, we require that, $\forall v \in V$, $C(v, v) = \{\text{Id}\}$, and that, $\forall v, v' \in V$, $C(v, v') = (C(v', v))^{-1}$. You can view an example QCN in Figure 1; for conciseness, converse relations or $\{\text{Id}\}$ loops are not shown in the figure.

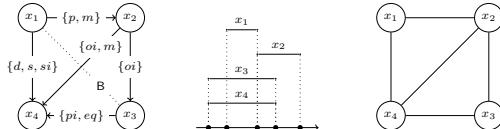


Figure 1: A QCN of Interval Algebra along with a solution, and a graph G

Let us also recall the definition of $\hat{\diamond}_G$ -consistency:

Definition 2. Given a QCN $\mathcal{N} = (V, C)$ and a graph $G = (V, E)$, \mathcal{N} is $\hat{\diamond}_G$ -consistent iff, $\forall \{v_i, v_j\}, \{v_i, v_k\}, \{v_k, v_j\} \in E$, $C(v_i, v_k) \subseteq C(v_i, v_j) \diamond C(v_k, v_j)$

Intuitively, $\hat{\diamond}_G$ -consistency checks if all triples of variables in \mathcal{N} that correspond to triangles in G are closed under weak composition.

Remember also that \diamond -consistency is $\hat{\diamond}_G$ -consistency where G is a complete graph. Hence, \diamond -consistency can be viewed as a special case of $\hat{\diamond}_G$ -consistency.

2 Exercices

Exercice 1. Construct a QCN of Interval Algebra for representing the temporal information of the following piece of text:

"In the morning, I have breakfast and power on my laptop to answer some emails. In general, my productive work takes place in the afternoon, and I usually have a lunch break around 13h00. In the evening, I like to prepare a nice dinner, and sometimes meet up with friends for a walk either before or after my meal."

Advice: Depending on what assumption(s) you make, you might get a different QCN in the end, and that is OK!

Exercice 2. Provide a formal proof for Proposition 1:

Proposition 1. The satisfiability checking problem and the minimal labeling problem of a QCN are equivalent under polynomial-time Turing reductions, i.e., the former is polynomial-time Turing reducible to the latter, and vice versa.

Let us first show that the satisfiability checking problem is polynomial-time Turing reducible to the minimal labeling problem. (This means that we have an oracle for solving the minimal labeling problem and we can use it a polynomial number of times to solve the satisfiability checking problem.) Let $\mathcal{N} = (V, C)$ be a QCN, and $u, v \in V$ any two of its variables. Then, for each base relation $b \in C(u, v)$ we construct a new QCN $\mathcal{N}_b = (V, C_b)$ such that:

- $C_b(u, v) = \{b\}$;
- $C_b(v, u) = \{b\}^{-1}$;
- and, $\forall (u', v') \in (V \times V) \setminus \{(u, v), (v, u)\}, C_b(u', v') = C(u', v')$.

In other words, each QCN \mathcal{N}_b is the original QCN \mathcal{N} where the base relation b has been assigned to the constraint $C(u, v)$. For each such \mathcal{N}_b , we ask our oracle to decide whether $C_b(u, v)$ contains unfeasible/impossible base relations (i.e., base relations that cannot be satisfied by any solution of \mathcal{N}_b). If the oracle returns NO for some \mathcal{N}_b , then the original QCN \mathcal{N} is satisfiable; otherwise, if the oracle returns YES for every \mathcal{N}_b , then the original QCN \mathcal{N} is unsatisfiable.

Next, we need to show that the minimal labeling problem is polynomial-time Turing reducible to the satisfiability checking problem. We follow the same construction of the set of QCNs \mathcal{N}_b as before, and we use the (satisfiability checking) oracle as follows. If the oracle returns NO for some \mathcal{N}_b , then the constraint $C(u, v)$ (and the original QCN \mathcal{N}) is not minimal, i.e., it contains unfeasible base relations; otherwise, if the oracle returns YES for every \mathcal{N}_b , then the constraint $C(u, v)$ is minimal (but we do not know whether the entire original QCN \mathcal{N} is minimal too; see the next exercise for an answer to this question).

Exercice 3. We say that a QCN $\mathcal{N} = (V, C)$ is minimal if and only if, $\forall u, v \in V$ and $\forall b \in C(u, v)$, b is a feasible base relation, i.e., a base relation that appears in some scenario of \mathcal{N} . Let \mathcal{N}_{\min} denote the unique equivalent minimal sub-QCN of a QCN \mathcal{N} :

1. Design an algorithm to compute \mathcal{N}_{\min} for a given QCN \mathcal{N} , provided that you have an oracle for solving the satisfiability checking problem of \mathcal{N} . Hint: Exploit your proof of Proposition 1.
2. Study the computational time complexity of your algorithm, provided that the aforementioned oracle runs in α time.

1. As hinted, to compute \mathcal{N}_{\min} we can perform the steps of the second part of the proof of Exercise 2, for every constraint in \mathcal{N} . Every time that the satisfiability checking oracle returns NO for the QCN that results by assigning a base relation to a constraint, we do not include that base relation in \mathcal{N}_{\min} . In the end, it should be clear that \mathcal{N}_{\min} will only contain base relations (in every constraint) that are feasible.

2. We need to call our oracle $O(|B| \cdot |V|^2)$ times, as we have $O(|V|^2)$ constraints in the QCN and each of these contains $O(|B|)$ base relations. Since the oracle runs in α time, the total runtime is $O(\alpha \cdot |B| \cdot |V|^2)$.

Exercice 4. Given two QCNs $\mathcal{N}_1 = (V, C_1)$ and $\mathcal{N}_2 = (V, C_2)$, let $\mathcal{N}_2 \cup \mathcal{N}_2$ denote the QCN (V, C) where, $\forall u, v \in V, C(u, v) = C_1(u, v) \cup C_2(u, v)$. Provide a formal proof for Proposition 2:

Proposition 2. Let $\mathcal{N}_1 = (V, C_1)$ and $\mathcal{N}_2 = (V, C_2)$ be any two QCNs, and $G = (V, E)$ any graph. Then, if \mathcal{N}_1 and \mathcal{N}_2 are $\hat{\diamond}_G$ -consistent, so is $\mathcal{N}_1 \cup \mathcal{N}_2$.

Exercice 5. Explain how the result of Proposition 2 can be used to assert that, for any QCN $\mathcal{N} = (V, C)$ and any graph $G = (V, E)$, there exists a unique largest $\hat{\diamond}_G$ -consistent sub-QCN of \mathcal{N} , i.e., a closure of \mathcal{N} under $\hat{\diamond}_G$ -consistency (in the course we denoted this closure by $\hat{\diamond}_G(\mathcal{N})$; see again the dominance property).

Exercice 6. Consider the QCN of Interval Algebra and the graph G in Figure 1, and answer the following questions:

1. Check whether the QCN is $\hat{\diamond}_G$ -consistent and detail your steps; if it is not $\hat{\diamond}_G$ -consistent, apply algebraic closure under $\hat{\diamond}_G$ -consistency to make it $\hat{\diamond}_G$ -consistent and detail your steps.
2. Check whether the QCN is \diamond -consistent and detail your steps; if it is not \diamond -consistent, apply algebraic closure under \diamond -consistency to make it \diamond -consistent and detail your steps.
- Subquestion: Can we exploit the outcome of Question 1, and to what extent?
3. Replace edge $\{x_2, x_4\}$ with edge $\{x_1, x_3\}$ in graph G , and repeat Question 1.

4. Try to obtain a scenario of the QCN and detail your steps.

1. To check whether the QCN is \circ_G -consistent or not, we just follow the definition of \circ_G -consistency (see the course slides). In sum, we need to check if all triples of variables of the QCN that correspond to triangles in the given graph G are closed under (weak) composition. In this case, we have two such triples of variables, viz., $\{x_1, x_2, x_4\}$ and $\{x_1, x_3, x_4\}$. Please note that for each such triple we need to perform a closure check for each of its three constraints. For example, let us start with $\{x_1, x_2, x_4\}$. Then, we need to perform the three following closure checks:

- $C(x_1, x_2) \subseteq C(x_1, x_4) \diamond C(x_4, x_2)$;
- $C(x_1, x_4) \subseteq C(x_1, x_2) \diamond C(x_2, x_4)$;
- and $C(x_2, x_4) \subseteq C(x_2, x_1) \diamond C(x_1, x_4)$.

Please note that it does not matter if we choose a constraint $C(u, v)$, or if we choose its converse $C(v, u)$, for a closure check, as we are dealing with relation algebras, which come with certain nice properties (see the course slides). Let us perform the closure check for $C(x_1, x_4) \subseteq C(x_1, x_2) \diamond C(x_2, x_4)$. We have that $C(x_1, x_2) \diamond C(x_2, x_4) = \{p \diamond oi\} \cup \{p \diamond m\} \cup \{m \diamond oi\} \cup \{m \diamond m\} = \{p, d, s, m, o\} \cup \{p\} \cup \{d, s, o\} \cup \{p\} = \{p, d, s, m, o\} \not\supseteq C(x_1, x_4) = \{d, s, si\}$. Thus, the QCN is not \circ_G -consistent, as a closure check failed for one of our triples of variables.

To apply \circ_G -consistency, we just iteratively perform the following operation for all triples of variables of the QCN that correspond to triangles in the given graph G until a fixed state is reached:

$$\forall \{v_i, v_j\}, \{v_i, v_k\}, \{v_k, v_j\} \in E, C(v_i, v_j) \leftarrow C(v_i, v_j) \cap ((v_i, v_k) \diamond C(v_k, v_j))$$

For example, earlier we saw that $C(x_1, x_4) = \{d, s, si\}$, but, on the other hand, $C(x_1, x_2) \diamond C(x_2, x_4) = \{p, d, s, m, o\}$. So, $C(x_1, x_4)$ should be updated to $\{d, s, si\} \cap \{p, d, s, m, o\} = \{d, s\}$. Please note that every time a constraint gets updated, we need to propagate the changes until we reach a fixed state where no more propagation is possible. To this end, there is a naive and a state-of-the-art algorithm, as explained in the course, and you may choose either of them for applying \circ_G -consistency. For instance, if we follow the state-of-the-art approach, since $C(x_1, x_4)$ got updated, we need to check whether the neighboring constraints $C(x_1, x_2)$ and $C(x_2, x_4)$ are affected. Indeed, $C(x_2, x_4)$ is affected, because $C(x_2, x_1) \diamond C(x_1, x_4) = \{d, f, mi, oi, pi\} \not\supseteq C(x_2, x_4) = \{oi, m\}$. So, $C(x_2, x_4)$ should be updated to $\{d, f, mi, oi, pi\} \cap \{oi, m\} = \{oi\}$. Repeating these operations until a fixed state is reached, we will also have that $C(x_3, x_4)$ gets updated to $\{eq\}$. At this point, the QCN will have become \circ_G -consistent.

2. As explained in the course, \diamond -consistency is \circ_G -consistency where G is the complete graph on the set of variables of a given QCN. With respect to our QCN, this means that we must also take into account the edge $\{x_1, x_3\}$ in our \circ_G -consistency definition. It is clear that we can use the refined, \circ_G -consistent QCN of the previous step as a starting point, and refine it further (if necessary).

Indeed, we can see that the QCN is not \diamond -consistent and it needs to be refined further. Specifically, we have that $C(x_1, x_4) \diamond C(x_4, x_3) = \{d, s\} \not\supseteq C(x_1, x_3) = B$ (remember that B is the set of all base relations, viz., the universal relation). So, $C(x_1, x_3)$ should be updated to $\{d, s\} \cap B = \{d, s\}$. At this point we can verify that the QCN is \diamond -consistent as no more propagation is possible.

Exercice 7. Consider Point Algebra, where $B = \{<, =, >\}$, and find $r, s, t \in 2^B$ such that (weak) composition does not distribute over non-empty intersection, i.e., $r \diamond (s \cap t) \neq (r \diamond s) \cap (r \diamond t)$; before exploring cases, argue first whether it is possible to find such r, s, t or not.

3 Project (Optional)

Extend your implementation from the previous TD to include operations for converse and (weak) composition. Specifically, build upon your abstract data type for representing relations for an arbitrary set of JEPD base relations B , and implement methods to compute converse and (weak) composition for such relations. At the end of this assignment you should have implemented methods that allow computations like $(\{<, =\})^{-1} \diamond \{>, <\}$ to be performed.

Representing QCNs Further extend your implementation to represent QCNs of some qualitative constraint language. Specifically, you should be able to read a QCN definition from a file with the following format:

```
n #description
x1 x2 rel1
x1 x3 rel2
.
```

Some details follow:

- n represents the total number of variables, **description** is for documentation only.
- $x1, x2, \dots$ are non-negative integers in the range $[0, n - 1]$, representing the n variables.
- **rel1, rel2, ...** are relations represented as whitespace-separated lists inside parentheses, e.g., $(< =)$.
- All constraints not defined are assumed to be the universal relation, converses *may* appear.

Supporting \diamond -consistency Finally, implement a method for checking the \diamond -consistency of an input QCN.

Qualitative Constraint Network: Reasoning

Travaux dirigés du cours *Contraintes* (HAI910I)

Michael Sioutis

27 Nov 2023

1 Graph Concepts

As a reminder from the course, we use $G(\mathcal{N})$ to denote the *constraint graph* (V, E) of a QCN $\mathcal{N} = (V, C)$, where $\{v, v'\} \in E$ iff $C(v, v') \neq \text{B}$. In other words, the constraint graph of a QCN corresponds to the part of the QCN that involves constraints, i.e., non-universal relations.

Let us now recall the definition of a chordal graph:

Definition 1. Let $G = (V, E)$ be an undirected graph, then G is chordal (or triangulated) if every cycle of length greater than 3 has a chord, which is an edge connecting two non-adjacent nodes of the cycle

Next, we view the relationship between a chordal graph and a tree decomposition:

Theorem 1. A graph G is chordal if and only if it has a tree decomposition $(T, \{X_1, \dots, X_n\})$ where cluster X_i is a clique of G for every $i \in \{1, \dots, n\}$.

A tree decomposition is formally defined as follows:

Definition 2. A tree decomposition of a graph $G = (V, E)$ is a tuple (T, X) where $T = (I, F)$ is a tree and $X = \{X_i \subseteq V \mid i \in I\}$ is a collection of clusters (subsets of V) that satisfies the following conditions:

1. For every $v \in V$ there is at least one node $i \in I$ such that $v \in X_i$.
2. For every $(u, v) \in E$ there exists a node $i \in I$ such that both $u, v \in X_i$.
3. Let i_1, i_2, i_3 be three nodes in I such that i_2 lies on the (unique) path between i_1 and i_3 in T . Then, if $v \in V$ belongs to both X_{i_1} and X_{i_3} , v must also belong to X_{i_2} .

Let us view the example presented in Figure 1. In the upper part of the figure we can view a graph $G = (V, E)$ (which can correspond to the constraint graph $G(\mathcal{N}) = (V, E)$ of a QCN \mathcal{N}). For the moment, we consider only the solid edges to be part of G and we disregard the dashed edges $\{3, 4\}$ and $\{4, 5\}$.

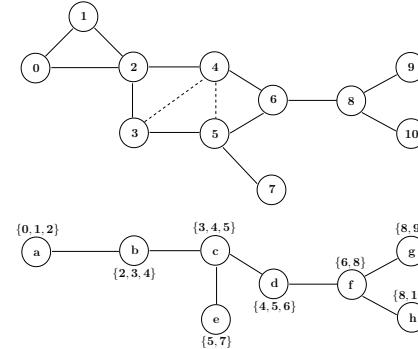


Figure 1: A graph (upper part) and its tree decomposition (lower part)

A tree decomposition of G comprises a tree $T = (I, F)$ with the set of nodes $I = \{a, b, c, d, e, f, g, h\}$ and a cluster X_i for every node $i \in I$ of that tree, as shown in the lower part of the figure, e.g., $X_a = \{0, 1, 2\}$. The first two conditions in our definition state that G is the union of the subgraphs induced by X_i , for every $i \in I$. The third condition implies that these subgraphs are organized roughly like a tree. Now, if we include the dashed edges $\{3, 4\}$ and $\{4, 5\}$ in G , then the clusters of the tree decomposition become *cliques*, namely, sets of vertices such that every two vertices in a set are connected by an edge; it is easy to verify that G is a chordal graph (as ensured by Theorem 1).

2 Exercices

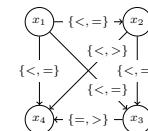


Figure 2: A QCN of Point Algebra

Exercice 1. Consider the QCN of Point Algebra in Figure 2, and answer the following questions:

1. Check whether the QCN is \diamond -consistent¹ and detail your steps; if it is not \diamond -consistent, apply algebraic closure under \diamond -consistency to make it \diamond -consistent and detail your steps.
2. Check whether the QCN is satisfiable and detail your steps.
3. Check whether the QCN is minimal (recall the definition of a minimal QCN from the course and the TD of last week) and detail your steps.

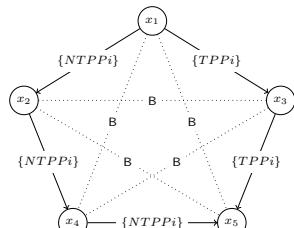


Figure 3: A QCN of RCC8

Exercice 2. Consider the QCN of RCC8 in Figure 3, and answer the following questions:

1. Provide the constraint graph of the QCN.
2. Check whether the constraint graph of the QCN is chordal and detail your steps; if it is not chordal, introduce a set of chords to make it chordal.
3. Check whether the QCN is \hat{G} -consistent w.r.t the chordal graph G of the previous step and detail your steps; if it is not \hat{G} -consistent, apply algebraic closure under \hat{G} -consistency to make it \hat{G} -consistent and detail your steps.
4. Check whether the QCN is satisfiable and detail your steps; specifically, detail the conditions and the procedures that allow you to decide if the QCN is satisfiable.
5. Produce a tree decomposition of the chordal graph G that you obtained in step 2, where every cluster is a clique of G .²

1. Based on our definition of the constraint graph of a QCN in the beginning of Section 1, the constraint graph here is the graph (V, E) where:

¹A reminder, \diamond -consistency is \hat{G} -consistency where G is the complete graph on the set of variables of a given QCN.

²This task serves to help you visualize how the different parts of the QCNs are patched together.

- $V = \{x_1, x_2, x_3, x_4, x_5\}$;
- and $E = \{\{x_1, x_2\}, \{x_1, x_3\}, \{x_2, x_4\}, \{x_3, x_5\}, \{x_4, x_5\}\}$.

Please note that a constraint graph is an undirected graph, i.e., $\{x_i, x_j\}$ is the same edge as $\{x_j, x_i\}$.

2. Based on the definition of a chordal graph in Section 1, the constraint graph of the QCN, viz., the graph (V, E) in the previous step, is not chordal, as variables $\{x_1, x_2, x_3, x_4, x_5\}$ form a cycle of length 5. There are many ways to make (V, E) chordal. In every case, we add a chord, i.e., an edge connecting two non-adjacent nodes (variables) of the cycle, and we check whether there still exists a cycle of length greater than 3. Here, we can start by introducing the chord $\{x_1, x_4\}$. Our graph will still not be chordal, as variables $\{x_1, x_3, x_4, x_5\}$ will be forming a cycle of length 4 in $(V, E \cup \{\{x_1, x_4\}\})$. If we also introduce the chord $\{x_1, x_5\}$, the graph will become chordal. We call this graph G . Specifically, $G = (V, E \cup \{\{x_1, x_4\}, \{x_1, x_5\}\})$. Please note that, by definition, a complete graph is also a chordal graph; so, one (bad) way of making a graph chordal is to make it complete by introducing all possible edges (i.e., by saturating the graph).

3. We have already seen how to check for and apply \hat{G} -consistency in the previous TD, so we will not detail this step. It suffices to say that in the end of the algebraic closure procedure, which will produce the unique largest³ \hat{G} -consistent sub-QCN of our QCN $\mathcal{N} = (V, C)$, the following updates will happen:

- $C(x_1, x_4)$ will be refined to $\{NTPPi\}$;
- and $C(x_1, x_5)$ will be refined to $\{TPPi, NTPPi\}$.

Clearly, these refinements are particular to the chordal graph that we produced in the previous step, and if we had produced a different chordal graph, we would have refined different constraints, possibly.

4. With respect to RCC8, and from our course notes, we can verify all of the following properties:

- our QCN is defined over a distributive and, hence, tractable subclass of relations of RCC8;
- all of the (maximal) tractable subclasses of relations of RCC8 have *patch-work*;
- our QCN is \hat{G} -consistent and not trivially inconsistent (i.e., it does not contain the empty set relation, viz., \emptyset);
- and the chordal graph G is a supergraph of the constraint graph of our QCN, i.e., $G \supseteq G(\mathcal{N})$.

From the proposition in slide 84 of our course notes, we can deduce that the QCN is satisfiable.

³w.r.t \subseteq

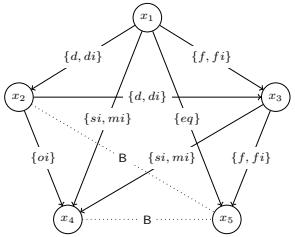


Figure 4: A QCN of Interval Algebra

Exercice 3. Consider the QCN of Interval Algebra in Figure 4, and answer the following questions:

1. Check whether the constraint graph of the QCN is chordal and detail your steps; if it is not chordal, introduce a set of chords to make it chordal.
2. Check whether the QCN is \hat{G} -consistent, where G is the chordal graph that you obtained in the previous step, and detail your steps; if it is not \hat{G} -consistent, apply algebraic closure under \hat{G} -consistency to make it \hat{G} -consistent and detail your steps.
3. Check whether the QCN is satisfiable and detail your steps; specifically, detail the conditions and the procedures that allow you to decide if the QCN is satisfiable.

3 Project (Optional)

Extend your implementation from the previous TD with methods to enforce \hat{G} -consistency and perform backtracking search. Basically, you will implement the two algorithms that were presented in the course. Note that you will have to make provision for a graph G to be provided as additional input to your solver.