Implémentation d'un solveur discret dans AbSolute

Mathieu Vavrille

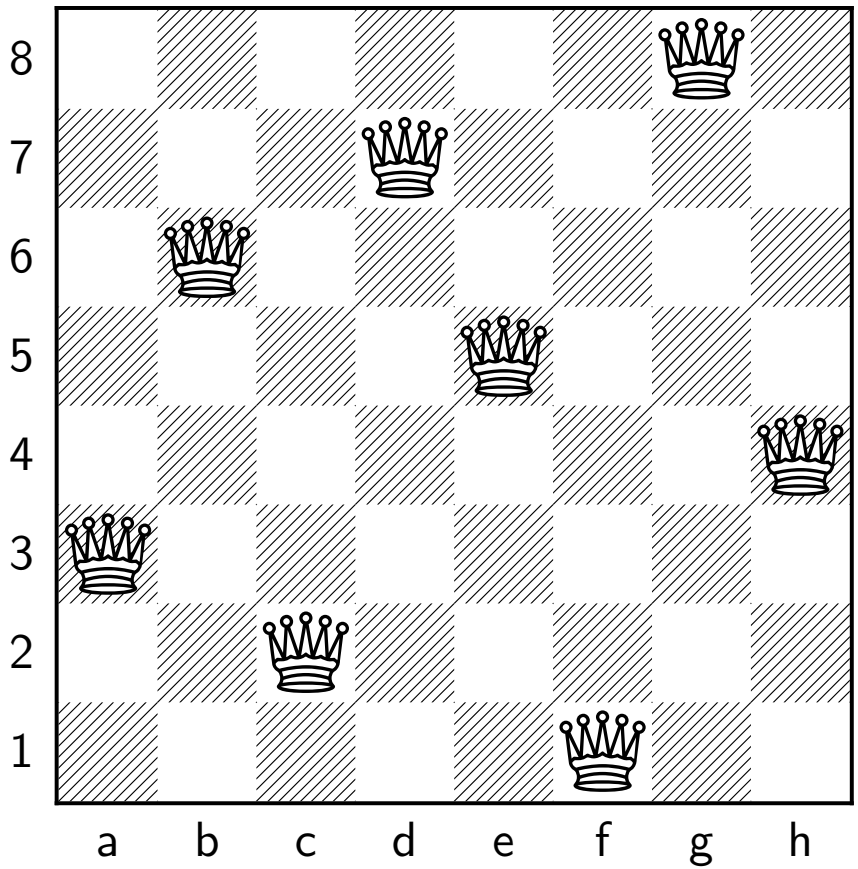
ENS de Lyon
Stage de L3 encadré par Charlotte Truchet
à l'Université de Nantes

13 Juillet 2017

UNIVERSITÉ DE NANTES

Introduction

$$C = (x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x1 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee x_4)$$

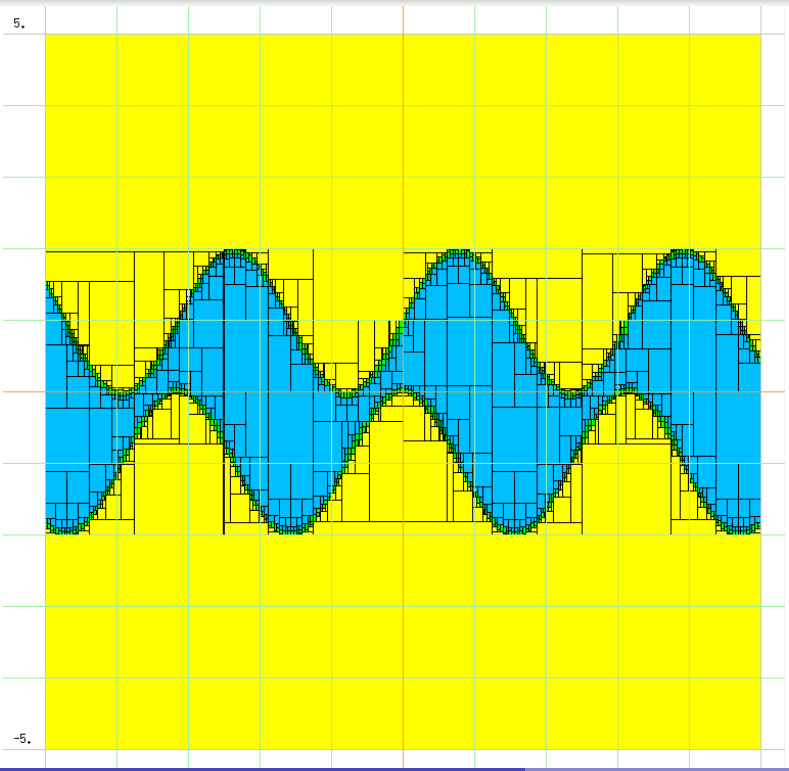


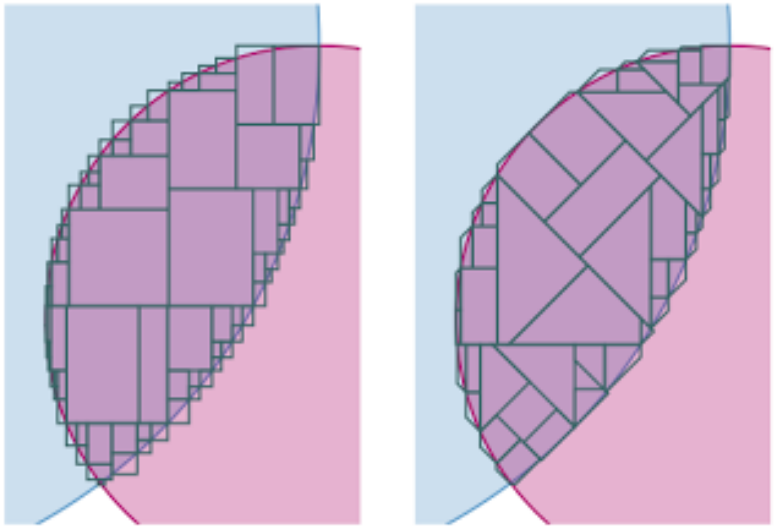
2			4	7	1			8
		8	3	6				
	7				9	4		2
1	5		9	8				
8	2						9	4
				4	5		8	7
7		2	5				1	
				2	6	7		
6			7	3	8			5

AbSolute

AbSolute [Pelleau et al., 2013]

- Solveur de contraintes, développé par Marie Pelleau, Ghilles Ziat, Antoine Miné, Charlotte Truchet
- Basé sur les domaines abstraits (de l'interprétation abstraite [Cousot and Cousot, 1977])
- Pour des variables réelles





1. Programmation par contraintes
   - Définitions
   - Algorithmes

2. Travail réalisé
   - Conception des domaines entiers
   - Implémentation dans AbSolute
   - Cadre général

1. Programmation par contraintes
   - Définitions
   - Algorithmes
2. Travail réalisé

Problème de satisfaction de contraintes

Définition (CSP)

Un problème de satisfaction de contraintes (CSP) est un triplet (X, D, C) où:

- $X = (x_1, \ldots, x_n)$ un ensemble de variables
- $D = (D_1, \ldots, D_n)$ les domaines des variables ($x_i \in D_i$)
- $C = (C_1, \ldots, C_m)$ un ensemble de contraintes

Une solution est une instanciation (affectation des valeurs à une variable) qui satisfait les contraintes.

Problème de satisfaction de contraintes

Définition (CSP)

Un problème de satisfaction de contraintes (CSP) est un triplet (X, D, C) où:

- $X = (x_1, \ldots, x_n)$ un ensemble de variables
- $D = (D_1, \ldots, D_n)$ les domaines des variables ($x_i \in D_i$)
- $C = (C_1, \ldots, C_m)$ un ensemble de contraintes

Une solution est une instanciation (affectation des valeurs à une variable) qui satisfait les contraintes.

But

- Trouver toutes les solutions (ou une seule)
- Trouver la solution minimisant une certaine fonction

Contraintes

Définition (Contraintes)

Soit $r \in \mathbb{N}$. Une contrainte est une relation définie sur les variables $X(c) = (x_{i_1}, \ldots, x_{i_r})$.
r est appelé arité de la contrainte et noté $|X(c)|$.

Contraintes

Définition (Contraintes)

Soit $r \in \mathbb{N}$. Une contrainte est une relation définie sur les variables $X(c) = (x_{i_1}, \ldots, x_{i_r})$.
r est appelé arité de la contrainte et noté $|X(c)|$.

Exemple

- $x + y = 3$
- $y * y + z * z > 10$
- $\max(y, x^5) < 200$
- $x < y \vee x > z$

Consistance

Difficulté

- Espace de recherche exponentiel
- Réduire les domaines des variables

Consistance

Difficulté

- Espace de recherche exponentiel
- Réduire les domaines des variables

Définition (Consistance d'arc)

Soit $c \in C$ et $x \in X(c)$
$v_i \in D(x_i)$ est consistante avec c si $\exists \tau \in c, \tau[x_i] = v_i$.
τ est appellé support pour v_i.
Un problème est arc-consistant si toutes ses valeurs sont consistantes.

Consistance

Difficulté

- Espace de recherche exponentiel
- Réduire les domaines des variables

Définition (Consistance d'arc)

Soit $c \in C$ et $x \in X(c)$
$v_i \in D(x_i)$ est consistante avec c si $\exists \tau \in c, \tau[x_i] = v_i$.
τ est appellé support pour v_i.
Un problème est arc-consistant si toutes ses valeurs sont consistantes.

Exemple

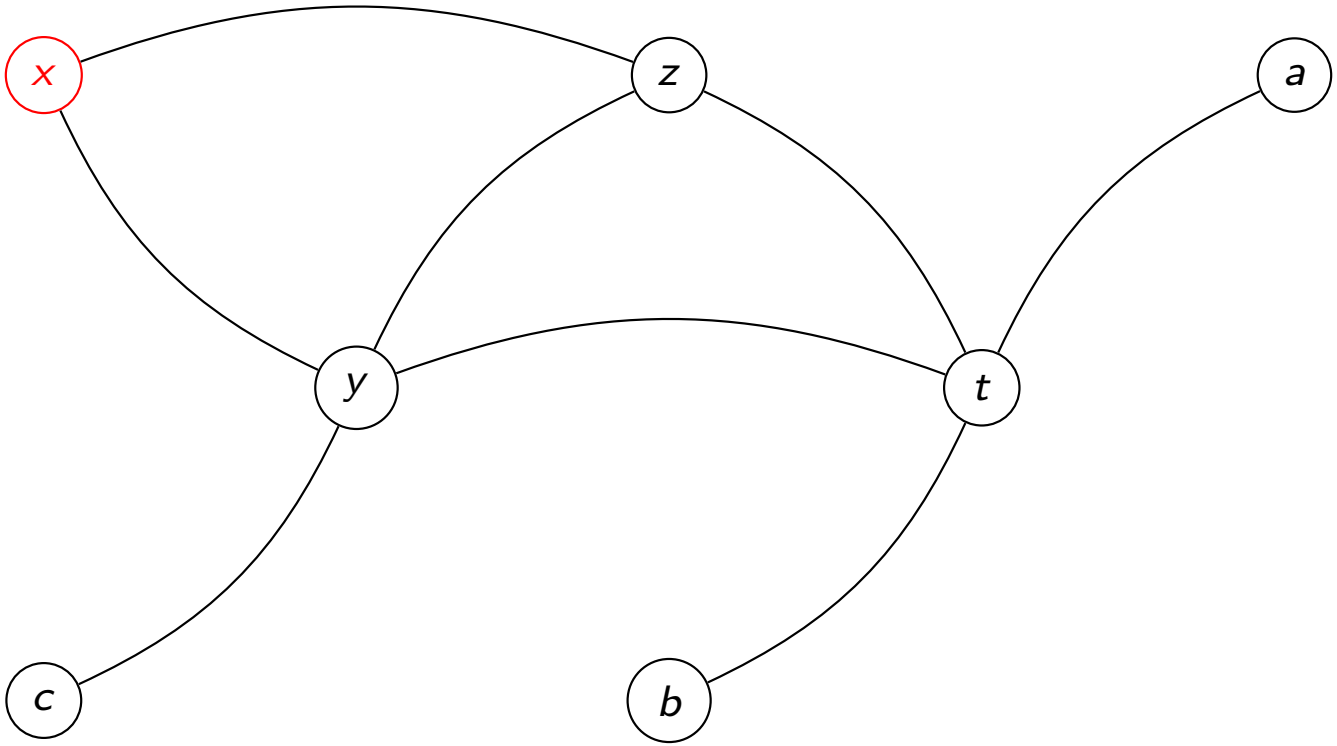
- $x \in [0, 10]$
- $y \in [0, 10]$
- $x + y \leq 5$

AC3 [Mackworth, 1977]

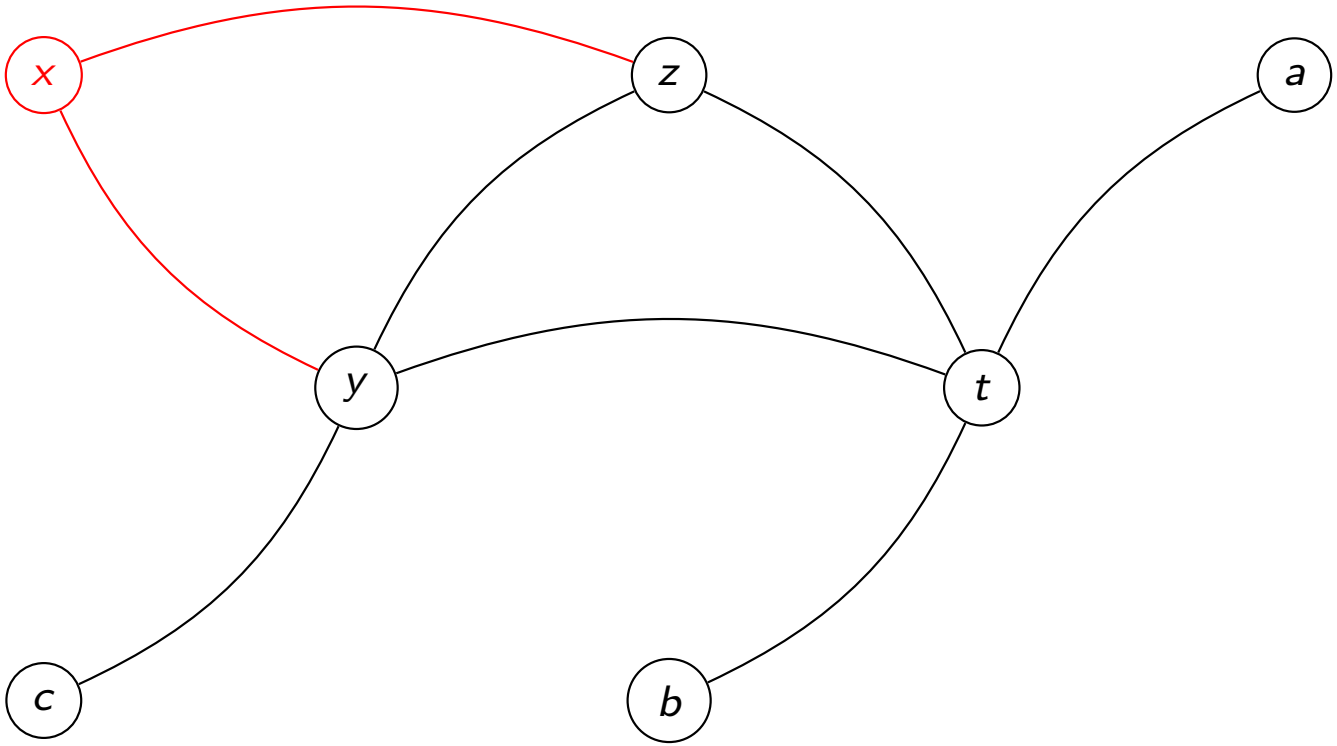
Consistance

function REVISE(x_i:variable, c:contrainte)
    CHANGE $\leftarrow$ *false*
    for $v_i \in D(x_i)$ **do**
        if v_i n'a pas de support pour c **then**
            Supprimer v_i de $D(x_i)$
            CHANGE $\leftarrow$ *true*
        end if
    end for
    return CHANGE
end function

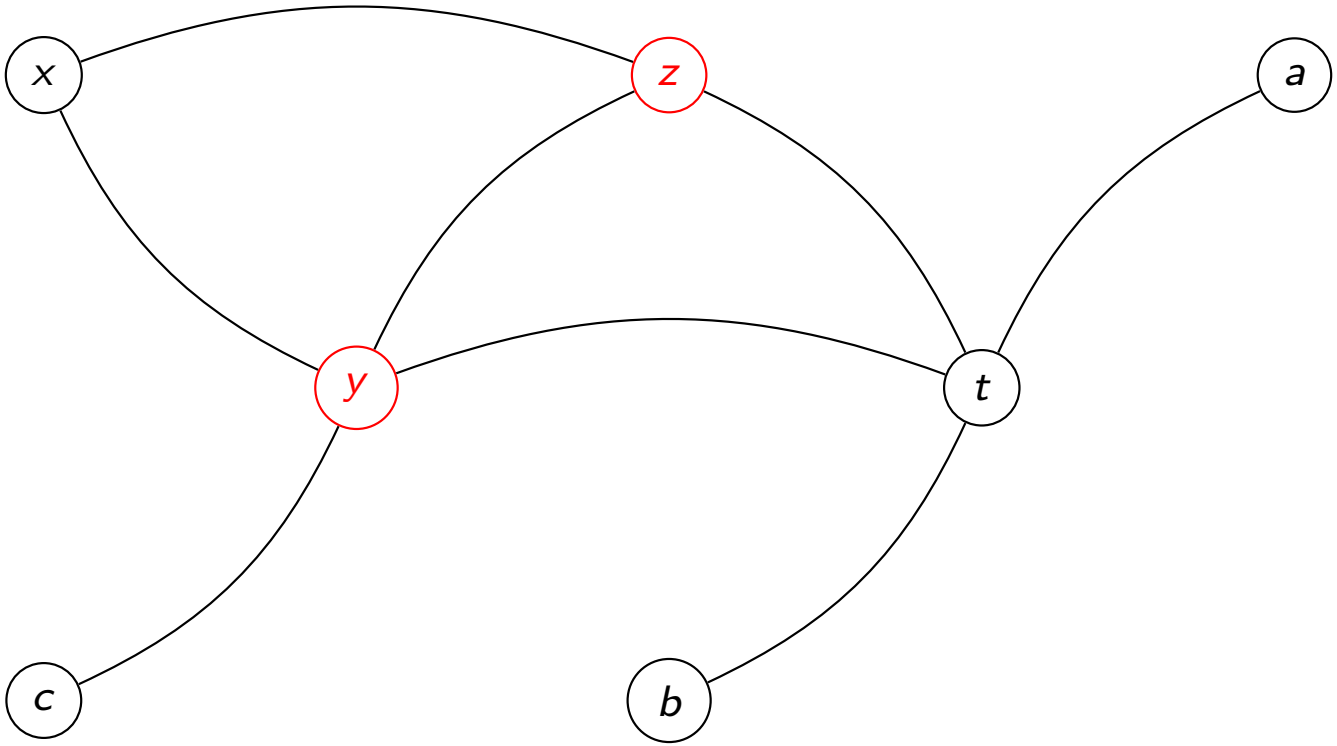
Propagation



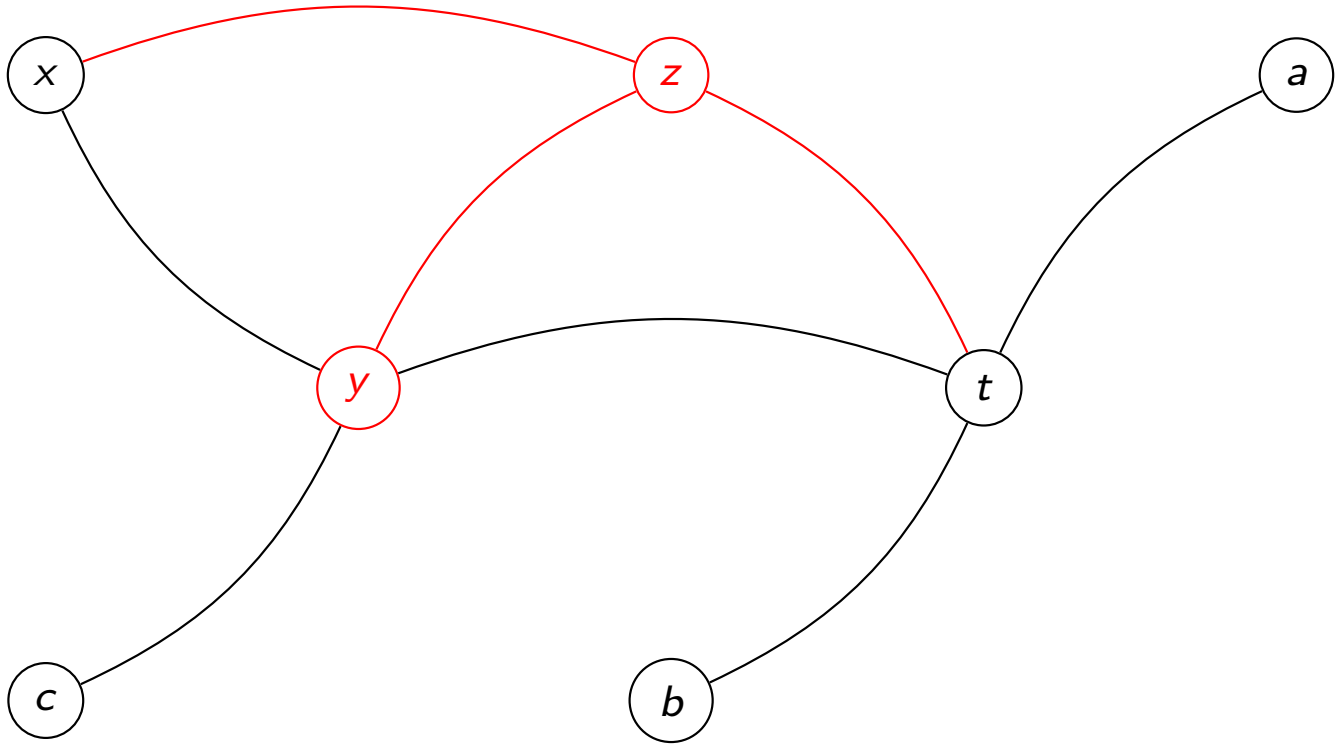
Propagation



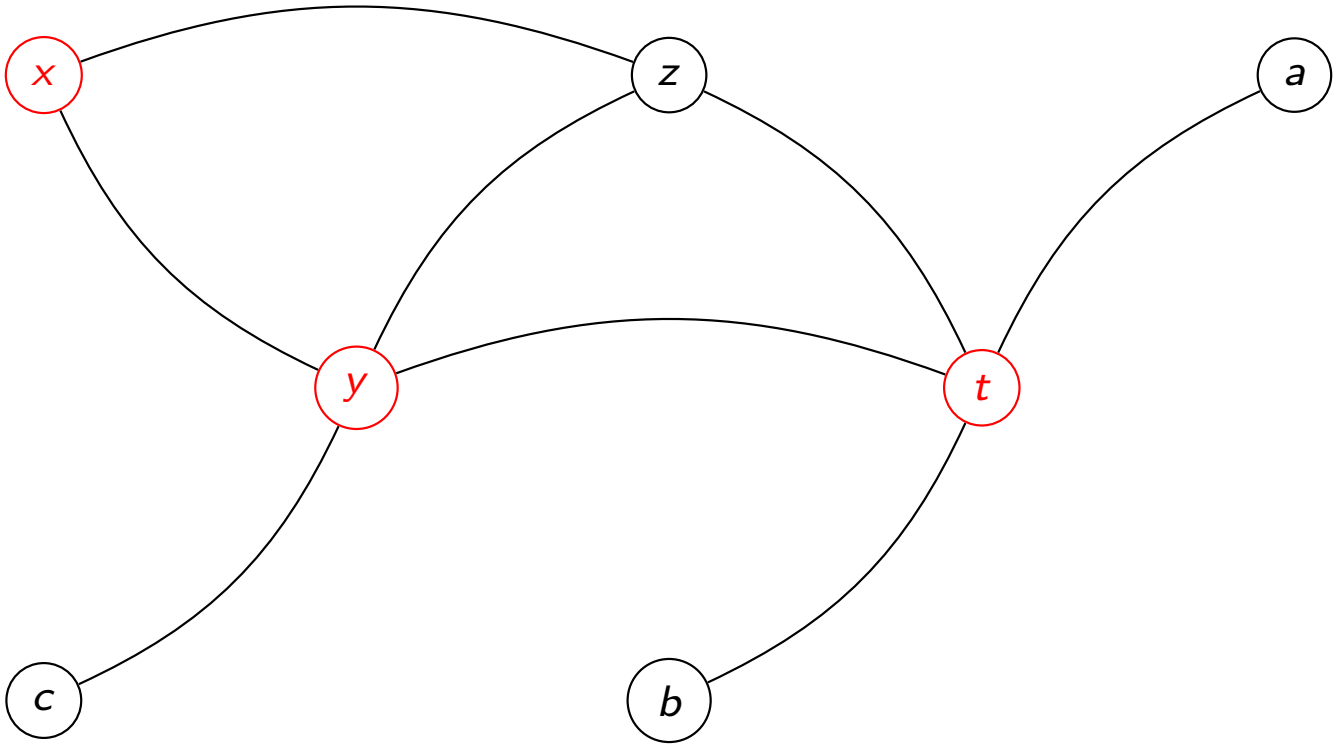
Propagation



Propagation



Propagation



AC3 [Mackworth, 1977]

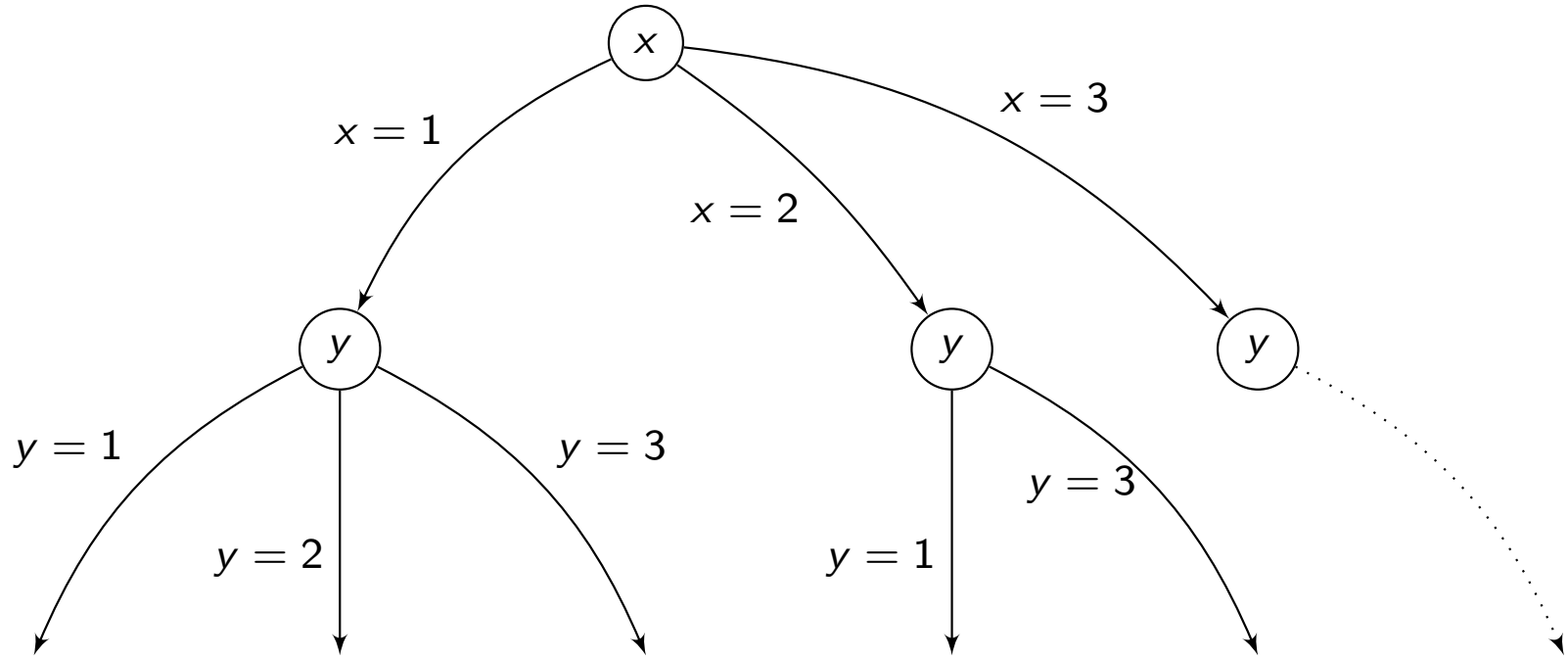
Propagation

function AC3((X, D, C): CSP)
  $Q \leftarrow \{(x_i, c) | c \in C, x_i \in X(c)\}$
  while $Q \neq \varnothing$ **do**
    $(x_i, c) \leftarrow pop(Q)$
    if *Revise*(x_i, c) **then**
      if $D(x_i) = \varnothing$ **then return** *false*
      else
        $Q \leftarrow Q \cup \{(x_j, c') | c' \in C, c' \neq c, x_i, x_j \in X(c), i \neq j\}$
      end if
    end if
  end while
  return *true*
end function

Recherche

Recherche

```
function BACKTRACK(P = (X, D, C): CSP)
    P ← PROPAGATION(P)
    if CONDITION_ARRET(P) ∧ IS_SOLUTION(P) then
        AFFICHER(P)
    else
        for csp in SPLIT(P) do
            BACKTRACK(csp)
        end for
    end if
end function
```


x
x = 1
x = 2
x = 3
y
y
y
y = 1
y = 2
y = 3
y = 1
y = 3

Améliorations [Rossi et al., 2006]

Consistance

Compromis entre vitesse et nombre de valeurs supprimées

- Consistance plus faibles (consistance de bornes)
- Consistance plus fortes (consistance de chemin, k-consistance,...)
- Consistances spécialisées (équations linéaires, contraintes globales, ...)

Améliorations [Rossi et al., 2006]

Consistance

Compromis entre vitesse et nombre de valeurs supprimées

- Consistance plus faibles (consistance de bornes)
- Consistance plus fortes (consistance de chemin, k-consistance,...)
- Consistances spécialisées (équations linéaires, contraintes globales, ...)

Propagation

- AC4: Se souvenir des supports [Mohr and Henderson, 1986]
- AC6: Se souvenir *d'un seul* support [Bessiere, 1994]
- AC2001: Stocker le *plus petit* support [Bessière et al., 2005]

All_different [Régin, 1994]

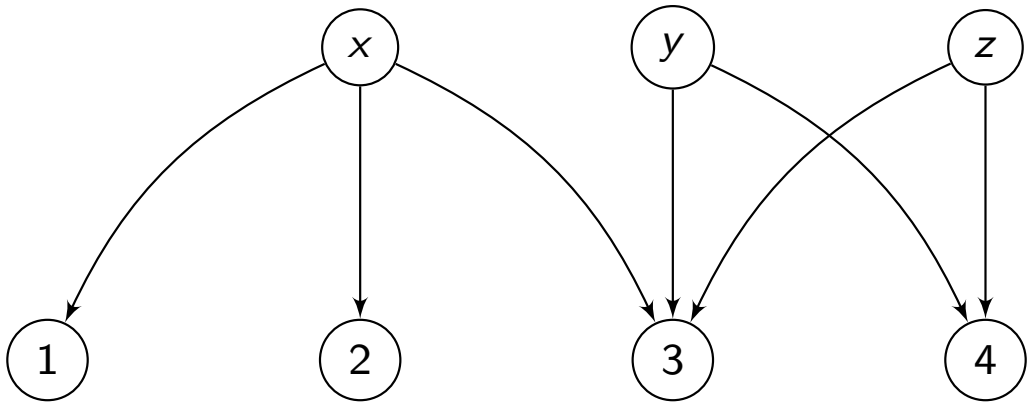
Définition

- $all\_different(x_1, \ldots, x_n) \equiv (\forall i < j, x_i \neq x_j)$
- Le graphe de valeurs est $G = (V, E)$ où $V = X \cup \bigcup_i D(x_i)$ et $E = \{(x_i, v_j) | x_i \in X, v_j \in D(x_i)\}$.

All_different [Régin, 1994]

Définition

- $all\_different(x_1, \ldots, x_n) \equiv (\forall i < j, x_i \neq x_j)$
- Le graphe de valeurs est $G = (V, E)$ où $V = X \cup \bigcup_i D(x_i)$ et $E = \{(x_i, v_j) | x_i \in X, v_j \in D(x_i)\}$.



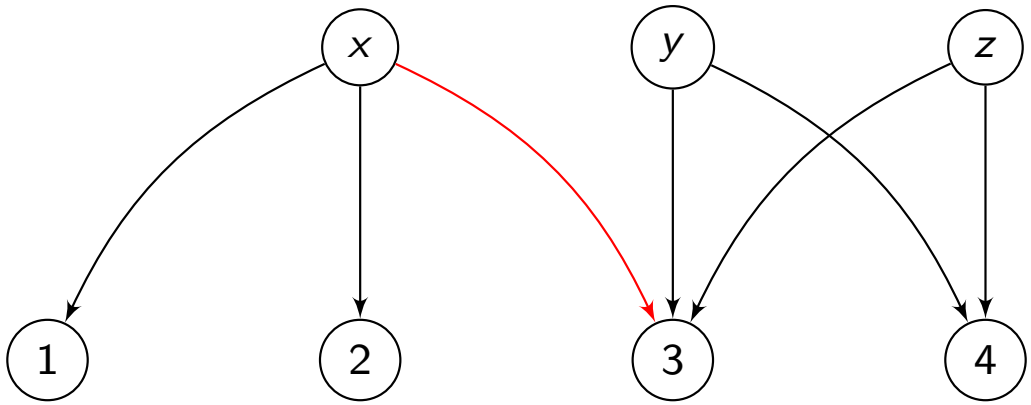
Théorème

Pour toute variable x_i, toute valeur $v_i \in D(x_i)$ est consistante ssi (x_i, v_i) apparaît dans un couplage couvrant X dans G.
Complexité: $O(\sqrt{|V|}|E|)$, et $O(|V| + |E|)$ incrémentalement.

All_different [Régin, 1994]

Définition

- $all\_different(x_1, \ldots, x_n) \equiv (\forall i < j, x_i \neq x_j)$
- Le graphe de valeurs est $G = (V, E)$ où $V = X \cup \bigcup_i D(x_i)$ et $E = \{(x_i, v_j) | x_i \in X, v_j \in D(x_i)\}$.



Théorème

Pour toute variable x_i, toute valeur $v_i \in D(x_i)$ est consistante ssi (x_i, v_i) apparaît dans un couplage couvrant X dans G.
Complexité: $O(\sqrt{|V|}|E|)$, et $O(|V| + |E|)$ incrémentalement.

1. Programmation par contraintes

2. Travail réalisé
   - Conception des domaines entiers
   - Implémentation dans AbSolute
   - Cadre général

Conception des domaines

Dans AbSolute

- Intégration dans le solveur
- Produit cartésien d'intervalles

Conception des domaines

Dans AbSolute

- Intégration dans le solveur
- Produit cartésien d'intervalles

Limitations

- Produit cartésien d'ensembles
- Réorganisation des contraintes
- Structure de données annexes

Conception des domaines

Dans AbSolute

- Intégration dans le solveur
- Produit cartésien d'intervalles

Limitations

- Produit cartésien d'ensembles
- Réorganisation des contraintes
- Structure de données annexes

Cadre général

- Structure de données spécifiques
- Consistances spécifiques

Structure d'AbSolute

x	y	...
		...

Table 1: Produit cartesien

- On associe à chaque valeur un *intervalle*

Structure d'AbSolute

x	y	...
[;]	[;]	...

Table 1: Produit cartesien

- On associe à chaque valeur un *intervalle*
- Intervalle formé de 2 *bornes*

Structure d'AbSolute

x	y	...
$[x_1; x_2]$	$[y_1; y_2]$	...

Table 1: Produit cartesien

- On associe à chaque valeur un *intervalle*
- Intervalle formé de 2 *bornes*
- Bornes sur lesquelles on sait faire des opérations

Travail effectué

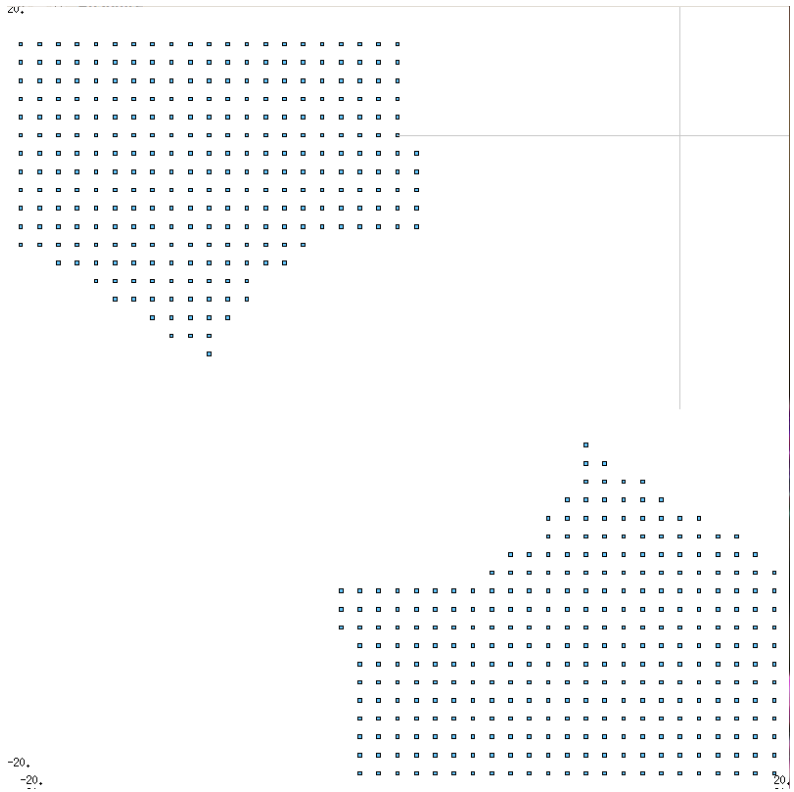
- Implémentation des intervalles entiers
  - avec les bornes entières: Problème des divisions
  - avec des rationnels (en calculs intermédiaires)

Travail effectué

- Implémentation des intervalles entiers
  - avec les bornes entières: Problème des divisions
  - avec des rationnels (en calculs intermédiaires)
- Amélioration du split entier: on énumère la liste des valeurs possibles

Travail effectué

- Implémentation des intervalles entiers
  - avec les bornes entières: Problème des divisions
  - avec des rationnels (en calculs intermédiaires)
- Amélioration du split entier: on énumère la liste des valeurs possibles
- Affichage des solutions



- $x \in [-20; 20]$
- $y \in [-20; 20]$
- $y * x + x * x/2 + y \leq 30$
- $y * y + x * x >= 100$

Amélioration des contraintes

Type de contraintes

- All_different
- Equation linéaire
- Inéquation linéaire
- Autre

Amélioration des contraintes

Type de contraintes

- All_different
- Equation linéaire
- Inéquation linéaire
- Autre

Pour "All_different", on exécute l'algorithme de consistance spécialisé.

Pour "Autre" on exécute la consistance d'AC4

Inéquation linéaire

Remarque [Codognet and Diaz, 1996]

On prend la contrainte $3 * x < 5 + 5 * z - y$. On la réécrit sous la forme:

- $3 * x < 5 + 5 * MAX(z) - MIN(y)$
- $y < 5 + 5 * MAX(z) - 3 * MIN(x)$
- $5 * z > -5 + 3 * MIN(x) + MIN(y)$

Inéquation linéaire

Remarque [Codognet and Diaz, 1996]

On prend la contrainte $3 * x < 5 + 5 * z - y$. On la réécrit sous la forme:

- $3 * x < 5 + 5 * MAX(z) - MIN(y)$
- $y < 5 + 5 * MAX(z) - 3 * MIN(x)$
- $5 * z > -5 + 3 * MIN(x) + MIN(y)$

Complexité

Pour la consistance: avec $n = |X(c)|$, on a un algo en $O(n^2)$

Equation linéaire

Remarque

Si $x \in \{-3; 1; 6\}, y \in \{-2; 3; 4\}$ alors
$2x + 3y + 3 \in \{-9; -1; 6; 9; 14; 17; 24; 27\} = E$
Les contraintes linéaires peuvent être rendues consistantes par intersection et difference d'ensemble.
Si $z = 2x + 3y + 3$ alors $D(z) \leftarrow D(z) \cap E$

- Égalité: $D(z) \leftarrow D(z) \cap E$
- Différence: Si $|E| = 1, D(z) \leftarrow D(z) \backslash E$

Conclusion

Ce qui a été fait

- Un solveur de contraintes discret
- Des propagations spécifiques

Continuation

- Intégration dans AbSolute
- Domaines mixtes (réels/discret)
- D'autres contraintes globales


Bessiere, C. (1994).
Arc-consistency and arc-consistency again.
Artificial intelligence, 65(1):179–190.


Bessière, C., Régin, J.-C., Yap, R. H., and Zhang, Y. (2005).
An optimal coarse-grained arc consistency algorithm.
Artificial Intelligence, 165(2):165–185.


Codognet, P. and Diaz, D. (1996).
Compiling constraints in clp (fd).
The Journal of Logic Programming, 27(3):185–226.


Cousot, P. and Cousot, R. (1977).
Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints.
In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM.


Mackworth, A. K. (1977).
Consistency in networks of relations.
Artificial intelligence, 8(1):99–118.


Mohr, R. and Henderson, T. C. (1986).
Arc and path consistency revisited.
Artificial intelligence, 28(2):225–233.


Pelleau, M., Miné, A., Truchet, C., and Benhamou, F. (2013).
A constraint solver based on abstract domains.
In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 434–454. Springer.


Régin, J.-C. (1994).
A filtering algorithm for constraints of difference in csps.
In *AAAI*, volume 94, pages 362–367.


Rossi, F., Van Beek, P., and Walsh, T. (2006).
Handbook of constraint programming.
Elsevier.

n-dames			
n	mode	temps	nb_noeuds
5	fast	0.01	22
	slow	0.7	22
6	fast	0.02	67
	slow	0.7	63
7	fast	0.04	180
	slow	2.1	168
8	fast	0.18	663
	slow	X	X
9	fast	0.45	2574
10	fast	2	10071
11	fast	10	43420
12	fast	48	207037

n-Langford			
n	mode	temps	nb_noeuds
4	fast	0.01	10
	slow	0.01	
5	fast	0.02	37
	slow	0.02	
6	fast	0.04	128
	slow	0.08	
7	fast	0.14	513
	slow	0.35	
8	fast	0.68	2592
	slow	2.1	
9	fast	3.8	14453
	slow	15	
10	fast	23	81818
11	fast	2m34	495195