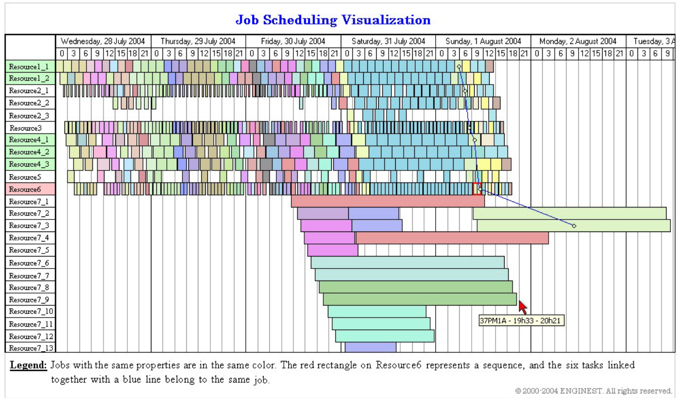
Programmation par contraintes dans des domaines finis

Alexander Bockmayr/Laurent Bougrain
LORIA / INRIA Nancy – Grand Est

UNIVERSITÉ DE LORRAINE

Exemples de problèmes industriels

Ordonnancement de tâches



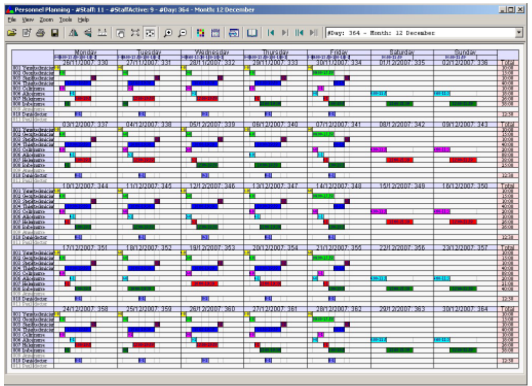
Objectifs :

- Diminution des ressources
- Diminution du temps d'exécution…

Contraintes :

- La tâche 1 nécessite la machine B pendant 24mn et doit être effectuée avant la tâche 7 et 12…

Emploi du temps



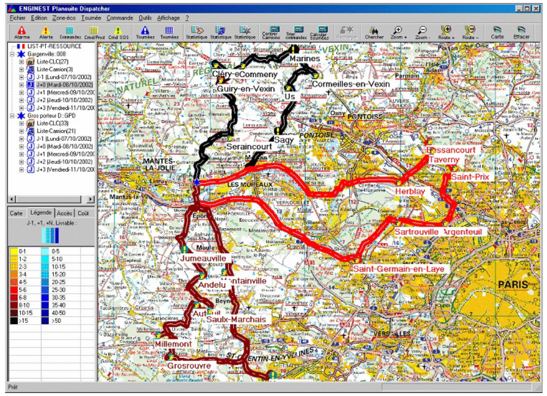
Objectifs :

- Diminuer les temps d'attente
- Diminuer le nombre de salles…

Contraintes :

- 1 technicien et 1 infirmier 24h/24
- 1 médecin en journée
- Réglementation social…

Tournées de véhicules



Objectifs :

- Diminution du kilométrage
- Diminution du nombre de véhicules Diminution du temps de travail…

Contraintes :

- Capacité des camions
- Disponibilité des véhicules…

Laurent.Bougrain@loria.fr

Bibliographie

- BOCKMAYR Alexander et HOOKER John : Constraint programming, in K. Aardal, G. Nemhauser and R. Weismantel, eds, Handbooks in OR & MS, vol. 12, Ed. Elsevier, 2004
- MACKWORTH Alan : Consistency in networks of relation, Artificial Intelligence, Vol 8, pp99-118, North-Holland Publishing compagny, 1977
- MACKWORTH Alan : Constraint satisfaction, S. Shapiro Ed. in Encyclopedia of artificial intelligence, pp285-293, Wiley, 1992
- SIMONIS Helmut, Constraint Logic Programming, 1995
- SOLNON Christine : cours de Programmation par Contraintes, 2003 http://www710.univ-lyon1.fr/~csolnon/Site-PPC/e-miage-ppc-som.htm

Laurent.Bougrain@loria.fr

Contraintes dans le domaine fini

Un problème de satisfaction de contraintes ou CSP (Constraint Satisfaction Problem) c'est :

- n variables x_1, x_2,…,x_n
- Pour chaque variable x_j, un domaine fini D_j de valeurs possibles (souvent D_j = N)
- m contraintes C_1, C_2,…,C_m où $C_i \subseteq D_{i1} x\ D_{i2}\ x \ldots x\ D_{ik1}$ est une relation entre k_i variables
- Une solution est l'affectation d'une valeur de D_j à x_j, pour tout j=1…n telle que toutes les relations C_i sont satisfaites

Laurent.Bougrain@loria.fr

Programmation par contraintes

- Idée de base : C'est une programmation qui inclue des contraintes, c'est-à-dire que le langage de programmation possède un solveur de contraintes.

- Contraintes : linéaire, non linéaire, orientée objet, booléenne…

- Programmation : logique, fonctionnelle, orientée objet, impérative, concurrente…

- Systèmes : Prolog III/IV, CHIP, ECLiPSe, ILOG, gprolog, Choco…

Laurent.Bougrain@loria.fr

Un problème de cryptarithmétique

Soit l'addition suivante :

```
  SEND
+ MORE
-----------
MONEY
```

où chaque lettre représente un chiffre différent (compris entre 0 et 9).

On souhaite connaître la valeur de chaque lettre, sachant que la première lettre de chaque mot représente un chiffre différent de 0.

1. Modéliser ce problème sous la forme d'un CSP.

Laurent.Bougrain@loria.fr

Un problème de cryptarithmétique : Solution 1

1. Modéliser ce problème sous la forme d'un CSP.

- Soit X = {S,E,N,D,M,O,R,Y} la liste des variables

- Soient D_S, D_E, D_N, D_D, D_M,D_O,D_R,D_Y les domaines respectivement de S,E,N,D,M,O,R,Y.
  On a:
  $D_S=D_M= \{1,\ldots,9\}$
  $D_E=D_N=D_D=D_O,D_R,D_Y= \{0, 1,\ldots,9\}$

- Soient les contraintes suivantes:
  C_1 : $1000*S+100*E+10*N+D + 1000*M+100*O+10*R+E = 10000*M+1000*O+100*N+10*E+Y$
  C_2 : $S \neq E$
  C_3 : $S \neq N$
  C_4 : $S \neq D$
  …
  C_{29} : $R \neq Y$

Laurent.Bougrain@loria.fr

Un problème de cryptarithmétique : Solution 2

1. Modéliser ce problème sous la forme d'un CSP.

- Soit X = {S,E,N,D,M,O,R,Y,R1,R2,R3} la liste des variables

- Soient D_S, D_E, D_N, D_D, D_M,D_O,D_R,D_Y,D_{R1},D_{R2},D_{R3} les domaines respectivement de S,E,N,D,M,O,R,Y,R1,R2,R3.
  On a:
  D_S=D_M= {1,…,9}
  D_E=D_N=D_D=D_O,D_R,D_Y= {0, 1,…,9}
  D_{R1},D_{R2},D_{R3} = {0,1}

- Soient les contraintes suivantes:
  C_1 : D+E = Y + 10*R1
  C_2 : R1+N+R = E + 10*R2
  C_3 : R2+E+O = N + 10*R3
  C_4 : R3+S+M = O +10*M
  C_5 : S ≠ E
  …
  C_{32} : R ≠ Y

Laurent.Bougrain@loria.fr

Le problème du zèbre

On s'intéresse au problème suivant, posé initialement par Lewis Caroll : *Cinq maisons consécutives, de couleurs différentes, sont habitées par des hommes de différentes nationalités. Chacun possède un animal différent, a une boisson préférée différente et fume des cigarettes différentes. De plus, on sait que :*

- *Le norvégien habite la première maison,*
- *La maison à coté de celle du norvégien est bleue,*
- *L'habitant de la troisième maison boit du lait,*
- *L'anglais habite la maison rouge,*
- *L'habitant de la maison verte boit du café,*
- *L'habitant de la maison jaune fume des kools,*
- *La maison blanche se trouve juste après la verte,*
- *L'espagnol a un chien,*
- *L'ukrainien boit du thé,*
- *Le japonais fume des cravens,*
- *Le fumeur de old golds a un escargot,*
- *Le fumeur de gitanes boit du vin,*
- *Le voisin du fumeur de Chesterfields a un renard,*
- *Le voisin du fumeur de kools a un cheval.*

1. Modéliser ce problème sous la forme d'un CSP.
2. *Qui boit de l'eau ?*
3. *A qui appartient le zèbre ?*

Laurent.Bougrain@loria.fr

Le problème du zèbre : solution

- *On définit le CSP (X,D,C) tel que*
- *Variables du problème: on associe une variable par attribut (couleur, animal, boisson, nationalité, cigarette) X = {blanche, rouge, verte, jaune, bleue, norvégien, anglais, ukrainien, japonais, espagnol, cheval, renard, zèbre, escargot, chien, thé, eau, lait, café, vin, kools, chesterfields, old_golds, cravens, gitanes}*
- *Domaines des variables: D(Xi) = {1,2,3,4,5}, pour toute variable Xi de X*
- *Contraintes:*
  - *On pose tout d'abord une contrainte pour chaque assertion de l'énoncé : norvégien = 1,*
    bleue = (norvégien + 1) ou (norvégien - 1)
    lait = 3,
    anglais = rouge,
    verte = café,
    jaune = kools,
    blanche = verte + 1,
    espagnol = chien,
    ukrainien = thé,
    japonais = cravens,
    old_golds = escargot,
    gitanes = vin,
    (chesterfields = renard + 1) ou (chesterfields = renard - 1),
    (kools = cheval + 1) ou (kools = cheval - 1)
  - *De plus, toutes les variables de même "type" doivent avoir des valeurs différentes (il ne peut pas y avoir plusieurs maisons qui ont la même couleur, ou un même animal, ...) blanche ≠ rouge ≠ verte ≠ jaune ≠ bleue,*
    thé ≠ eau ≠ lait ≠ café ≠ vin,
    norvégien ≠ anglais ≠ ukrainien ≠ japonais ≠ espagnol,
    cheval ≠ renard ≠ zèbre ≠ escargot ≠ chien ≠ thé,
    kools ≠ chesterfields ≠ old_golds ≠ cravens ≠ gitanes

Laurent.Bougrain@loria.fr

Problème d'affectation parcimonieuse

- Une école souhaite affecter ses étudiants à des stages en entreprise en respectant au mieux le choix des étudiants et des entreprises. Elle demande pour cela d'une part à chacun des étudiants de classer par ordre de préférence les stages qui les intéressent et d'autre part, aux entreprises de classer par ordre de préférence les étudiants qui les intéressent. Pour tenir compte au mieux des désirs des uns et des autres, ces listes peuvent être incomplètes. Par ailleurs, dans le cas où un étudiant indécis n'arrive pas à trancher entre plusieurs stages qui lui semblent également intéressants, il peut les classer ex aequo (idem pour les entreprises). Pour simplifier, on supposera que l'on a autant d'étudiants que d'entreprises. Il s'agit alors, à partir de ces listes de préférences éventuellement incomplètes et avec des ex aequo, de former des affectations stables. Par stable, on entend que personne ne refuse l'affectation proposée sous prétexte qu'on aurait pu faire mieux. Par exemple, si France est affectée à Philips alors qu'elle préférait France Télécom et si dans le même temps Philippe est affecté à France Télécom alors qu'il préférait Philips, nous sommes en présence d'une instabilité

Laurent.Bougrain@loria.fr

Problème d'affectation parcimonieuse

- Les classements des étudiants sont les suivants :
  - E1 préfère B2, puis (B6 et B4 ex aequo) ;
  - E2 préfère (B2, B5), B6 ;
  - E3 préfère B1, B3, B6 ;
  - E4 préfère B6, B3 ;
  - E5 préfère B2, B1, B5 ;
  - E6 préfère B6, (B4, B2), B5, B1.
- Les classements des entreprises sont les suivants :
  - B1 préfère (E5, E3), E6 ;
  - B2 préfère E2, E5, E1, E6 ;
  - B3 préfère (E3, E4) ;B4 préfère E6, E1 ;
  - B5 préfère E5, E2, E6 ;
  - B6 préfère E1, (E4, E6), E2, E3.

Q. 1 : Modéliser ce problème sous la forme d'un problème de satisfaction de contraintes (variables, domaines, contraintes).

Laurent.Bougrain@loria.fr

Problème d'affectation parcimonieuse : solution

- Variables

X ={boite-de-1, boite-de-2, boite-de-3, boite-de-4, boite-de-5, boite-de-6}

- Domaines :

Pour chaque variable boite-de-i, le domaine associé contient l'ensemble des entreprises qui sont classées par i et qui ont classées i :

  - D(boite-de-1) ={B2, B4, B6}
  - D(boite-de-2) ={B2, B5, B6}
  - D(boite-de-3) ={B1, B3, B6}
  - D(boite-de-4) ={B3, B6}
  - D(boite-de-5) ={B1, B2, B5}
  - D(boite-de-6) ={B1, B2, B4, B5, B6}

Il n'y a pas de problème particulier : toutes les entreprises choisies par les étudiants ont sélectionnées ces étudiants.

- Contraintes
  - C1 =∀i∈{1..6}∀j∈{1..6},j!=i, boite–de–i !=boite–de–j
  - C2 =∀i∈{1..6}∀j∈{1..6},j!=i, non (préfère(i,boite-de-j,boite-de-i) et préfère(boite-de-j, i, j))

Laurent.Bougrain@loria.fr

Résolution par génère et teste

- Instancier les variables (dans l'ordre de déclaration à défaut d'heuristique de sélection).
- Quand toutes les variables sont instanciées, déterminer si toutes les contraintes sont respectées.
- Si une contrainte n'est pas satisfaite, retourner à la dernière variable dont le domaine contient encore des valeurs non affectées.

Laurent.Bougrain@loria.fr

Résolution par backtracking

- Instancier les variables dans l'ordre.
- Dès que toutes les variables d'une contrainte sont instanciées, déterminer si la contrainte est respectée.
- Si la contrainte n'est pas satisfaite, retourner à la dernière variable dont le domaine contient encore des valeurs non affectées, sinon continuer l'instanciation.

Laurent.Bougrain@loria.fr

Problèmes d'efficacité

Mackworth 77

- Si le domaine D_i de la variable x_i contient une valeur v qui ne satisfait pas C_i, alors toutes les instanciations futures aboutiront à deséchecs.

- Si nous instancions les variables dans l'ordre x_1, x_2,…,x_n et si pour x_i = v il n'y a pas de valeur w $\in$ D_j, pour j > i, telle que C_{ij}(v, w) soit satisfaite, alors le backtracking essaiera toutes les valeurs de $x_j$échouera et essaiera toutes les valeurs de x_{j-1} puis toutes les valeurs de x_j de nouveau,échouera et ce jusqu'à ce qu'on essaie toutes les combinaisons de x_{i+1}, x_{i+2},…,x_j avant de découvrir finalement que la valeur v n'est pas possible pour x_i.

Laurent.Bougrain@loria.fr

Consistance locale

- Soit un CSP avec uniquement des contraintes unaires et binaires.
- Soit G le graphe des contraintes
  - Chaque variable x_i est représentée par un nœud i
  - Chaque paire de variable x_i,x_j apparaissant dans une même contrainte est représentée par deux arcs (i,j) et (j,i).

- Le nœud i est consistant si $C_i(v)$ est satisfait pour tout $v \in D_i$

- L'arc (i,j) est consistant si pour tout $v \in D_i$ satisfaisant $C_i(v)$ il existe $w \in D_j$ satisfaisant $C_j(w)$ telles que $C_{ij}(v,w)$ soit satisfaite.

  Remarque : le fait que l'arc (i,j) soit consistant n'implique pas nécessairement que l'arc (i,j) le soitégalement

- Le graphe des contraintes est consistant au niveau des nœuds (resp. des arcs) si tous les nœuds (resp. les arcs) sont consistants.

Laurent.Bougrain@loria.fr

L'algorithme Arc-Consistency [Mackworth, 77]

L'algorithme Arc-Consistency (AC-3)

pour i de 1 à n faire $D_i \leftarrow \{v \in D_i \mid C_i(v)\}$

$Q \leftarrow \{(i,j) \mid (i,j) \in arcs(G)\}$

tant que Q n'est pas vide faire

sélectionner et supprimer un arc (i,j) de Q

si REVISE(i,j) est vraie alors $Q \leftarrow Q \cup \{(k,i) \mid (k,i) \in arcs(G), k \neq i, k \neq j\}$

Procedure REVISE(i,j)

DELETE $\leftarrow$ Faux

pour toute valeur $v \in D_i$ faire

s'il n'y a pas de valeur $w \in D_j$ telle que $C_{ij}(v,w)$ soit satisfaite alors

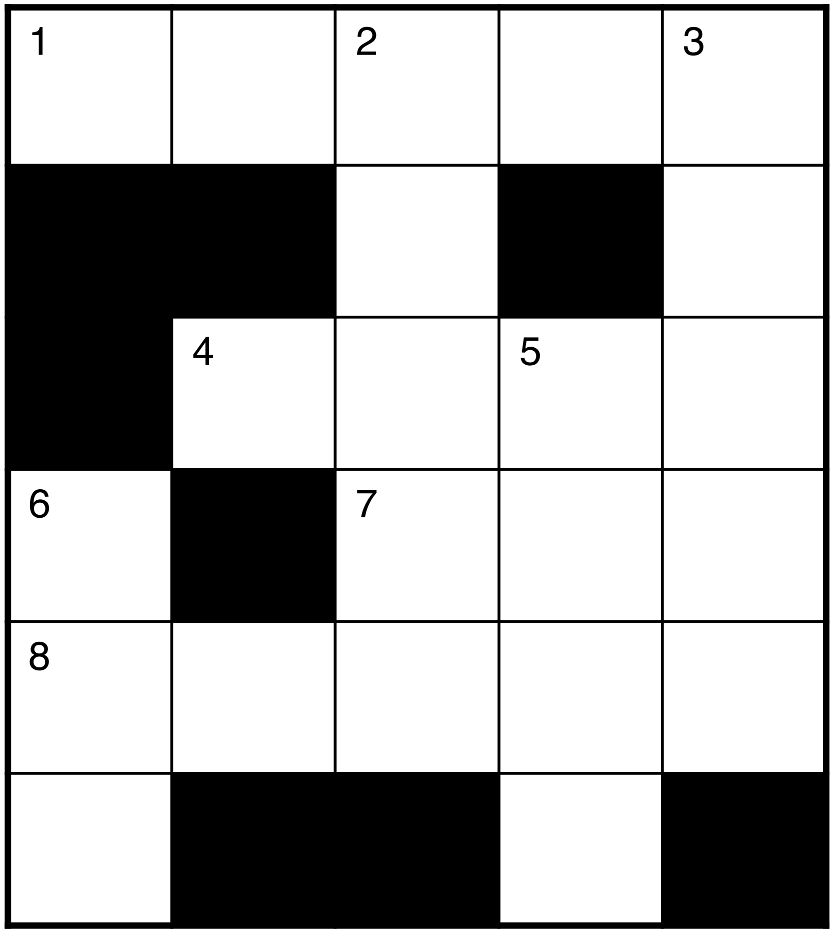
supprimer v de D_i

DELETE $\leftarrow$ Vrai

retourner DELETE

Laurent.Bougrain@loria.fr

Le problème de mots placés



Liste de mots

Aft
Ale
Eel
Heel
Hike
Hoses
Keel
Knot
Laser
Lee
Line
Sails
Sheet
Steer
Tie

1. Faire le graphe G des contraintes
2. Appliquer l'algorithme AC-3

Laurent.Bougrain@loria.fr

Les mots-placés : modélisation

- Chaque mot cherché est associé à une variable x_i
- Le domaine de définition de chaque variable est :

  D_i= {Aft, Ale, Eel, Heel, Hike, Hoses, Keel, Knot, Laser, Lee, Line, Sails, Sheet, Steer, Tie}
- On note C_i la contrainte unaire de chaque variable x_i : longueur(x_1)=5,…
- On note C_{ij} les contraintes binaires : $C_{12}=C(x_1,x_2)$ lettre(x_1,3)=lettre(x_2,1), …

1. Faire le graphe G des contraintes

Laurent.Bougrain@loria.fr

2. Appliquer l'algorithme AC-3

   D_1=D_2=D_3= {Hoses, Laser, Sails, Sheet, Steer}

   D_4=D_5= {Heel, Hike, Keel, Knot, Line}

   D_6=D_7= {Aft, Ale, Eel, Lee, Tie}

   D_8= {Hoses, Laser, Sails, Sheet, Steer}

   Q={(1,2),(2,1),(1,3),(3,1),(2,4),(4,2),(2,7),(7,2),(2,8),(8,2),(3,4),(4,3),(3,7),(7,3),(3,8),(8,3),(4,5),(5,4),(5,7),(7,5),(5,8),(8,5),(6,8),(8,6)}

   sélectionner (1,2),

   Q={(1,2),(2,1),(1,3),(3,1),(2,4),(4,2),(2,7),(7,2),(2,8),(8,2),(3,4),(4,3),(3,7),(7,3),(3,8),(8,3),(4,5),(5,4),(5,7),(7,5),(5,8),(8,5),(6,8),(8,6)}

   REVISE (1,2) <- true (D_1= {Hoses, Laser})

Q={(2,1),(1,3),(3,1),(2,4),(4,2),(2,7),(7,2),(2,8),(8,2),(3,4),(4,3),(3,7),(7,3),(3,8),(8,3),(4,5),(5,4),(5,7),(7,5),(5,8),(8,5),(6,8),(8,6)}

Laurent.Bougrain@loria.fr

Anticipations

Appliquer dynamiquement la consistance locale pendant la recherche.

- Forward checking : après l'affectation de la valeur v à x,éliminer pour toutes les variables y non encore instanciées les valeurs de D_y qui sont incompatibles avec v.

- Partial lookahead :établir la consistance des arcs pour tout couple (y, y') de variables non encore instanciées et où y sera instanciée avant y'.

- Full lookahead :établir la consistance des arcs pour tout couple (y, y') de variables non encore instanciées.

Laurent.Bougrain@loria.fr

Les 8 reines

- Chaque reine doit se trouver sur une ligne différente.
  Soit x_i la colonne de la reine qui se trouve sur la ligne i, i = 1,…,8
- Chaque domaine vaut $D_i = \{1,\ldots,8\}$
- Les contraintes binaires sont C_{ij} :
  $xi \neq xj$, pour $1 \leq i < j \leq 8$ (colonne)
  $xi \neq xj + (j - i)$, pour $1 \leq i < j \leq 8$ (diagonale 1)
  $xi \neq xj - (j - i)$, pour $1 \leq i < j \leq 8$ (diagonale 2)

Laurent.Bougrain@loria.fr

Les 8 reines (Forward checking)

Q							
X	X	Q					
X	X	X	X	Q			
X	Q	X	X	X	X		
X	X	X		X	X	X	
X	X	X	**X**	X	X	X	X
X	X	X		X		X	X
X	X	X		X	X		X

X_1=1

X_2=3

X_3=5

X_4=2,*7* *(plus de valeurs pour X_6)*

…

Laurent.Bougrain@loria.fr

Les 8 reines (Partial Lookahead)

Q							
X	X	Q					
X	X	X	X	Q			
X		X	X	X	X		
X		X		X	X	X	
X	X	X		X	X	X	X
X		X		X		X	X
X		X		X			X

$X_1=1$

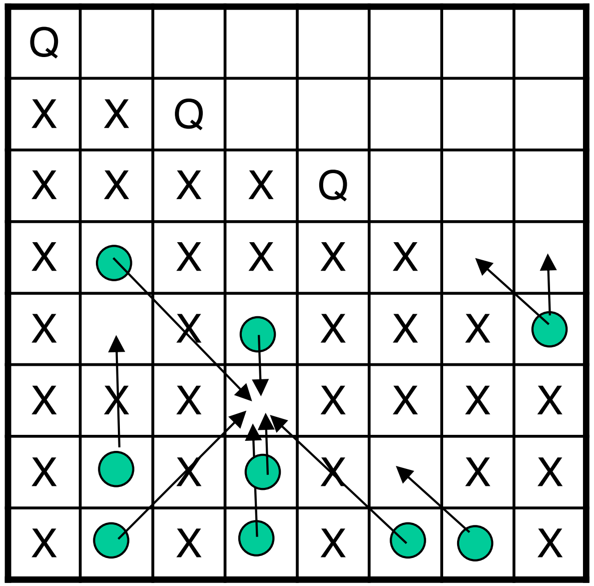
$X_2=3$

$X_3=5$ ($D_4= D_4 \setminus \{2\}$, $D_5= D_5 \setminus \{4\}$)

$X_4=7$

Laurent.Bougrain@loria.fr

Les 8 reines (Full Lookahead)



$X_1=1$

$X_2=3$

$X_3=5$ ($D_4= D_4\setminus\{2\}$, $D_5= D_5\setminus\{4\}$…)

$X_3=6$

Laurent.Bougrain@loria.fr

Struture classique d'un programme de satisfaction

- Déclaration des variables et de leur domaines de définitions

- Définition des contraintes

- Énumération (labeling)

Les solveurs de contraintes réalisent uniquement les consistances locales.

Dans le but d'obtenir les consistances globales, les domaines doivent êtreénumérés.

Laurent.Bougrain@loria.fr

Problème de cryptoarithmétique

- SEND+MORE = MONEY
- Attribuer à chaque lettre de {S,E,N,D,M,O,R,Y} un chiffre différent entre 0 et 9 tel que l'équation soit satisfaite.

```
Top([S,E,N,D,M,O,R,Y]) :-
        [S,E,N,D,M,O,R,Y]::0..9,
        alldifferent ([S,E,N,D,M,O,R,Y]),
        S #\= 0,
        M #\= 0,
        Send = 1000*S + 100*E + 10*N + D,
        More = 1000*M + 100*O + 10*R + E,
        Money = 10000*M + 1000*O + 100*N + 10*E + Y,
        Send + More #= Money,
        labeling ([S,E,N,D,M,O,R,Y]).

?- top(X)
X = [9,5,6,7,1,0,8,2]
```

Par propagation de contraintes, l'espace de recherche est réduit à S = 9, M = 1, O = 0, E : [4..7], N : [5..8], R, D, Y : [2..8] au noeud racine.

Un seul backtraking est nécessaire pour trouver une solution.

Laurent.Bougrain@loria.fr

Stratégies de sélection dynamique des variables

Par défaut, l'ordre d'instanciation des variables est celui de déclaration.
Des heuristiques peuvent permettre de diminuer l'arbre de recherche.

Nom	Stratégie
First_fail	variable avec le plus petit domaine [au moment de l'instantiation].
Most_constrained	variable qui apparaît dans le plus de contraintes et qui possède le plus petit domaine
Smallest (**largest**)	variable qui possède la valeur la plus petite (la plus grande)
Max_reget	variable qui a le plus grandécart entre sa valeur la plus petite et la suivante.

Laurent.Bougrain@loria.fr

Heuristiques d'énumération des valeurs

- **indomain(X)** : sélectionne les valeurs pour les variables X, en commençant par les plus petites
- **indomain(X, method)** : sélectionne les valeurs pour les variables X, en utilisant une des méthodes suivante : {min, max, middle}

- **Labeling(List,0,Method, Pred)** : Affecte des valeurs aux variables de la list List en utilisant
  - Method pour choisir la variable
  - Pred pour choisir la valeur

Exemple : labeling(List, 0, first_fail, indomain).

- **min_max(Goal,C)** : trouve une solution de Goal qui minimise le coût (maximal) de C.

Exemple : min_max(labelling(List,0,first_fail,indomain),C).

Laurent.Bougrain@loria.fr

Contraintes globales

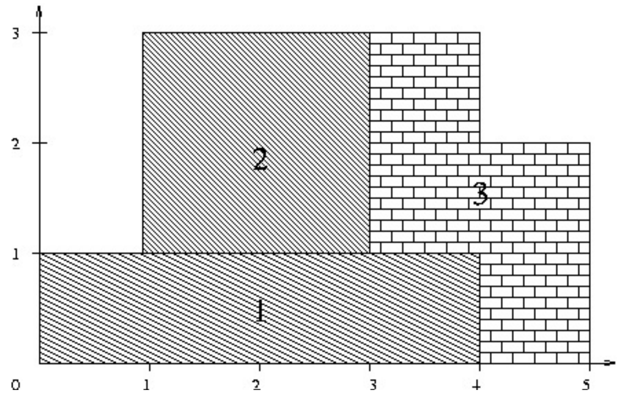
- **alldifferent**($[X_1,\ldots,X_n]$) : toutes les variables doivent prendre des valeurs différentes

i.e. $X_1 \neq \ldots \neq X_n$

Exemples: cryptoarithmétique, voyageur de commerce…

- **cumulative**($[S_1,\ldots,S_n]$, $[D_1,\ldots,D_n]$, $[R_1,\ldots,R_n]$,L) avec
  - n le nombre de taches,
  - S_i l'horaire de démarrage de la tache i,
  - D_i la durée de la tache i,
  - R_i le nombre de resources demandées par la tache I à chaque instant
  - L le nombre maximum de resources disponibles à un instant donné

Exemples: cumulative([0,3,1], [4,2,2], [1,2,2], 3)



Laurent.Bougrain@loria.fr

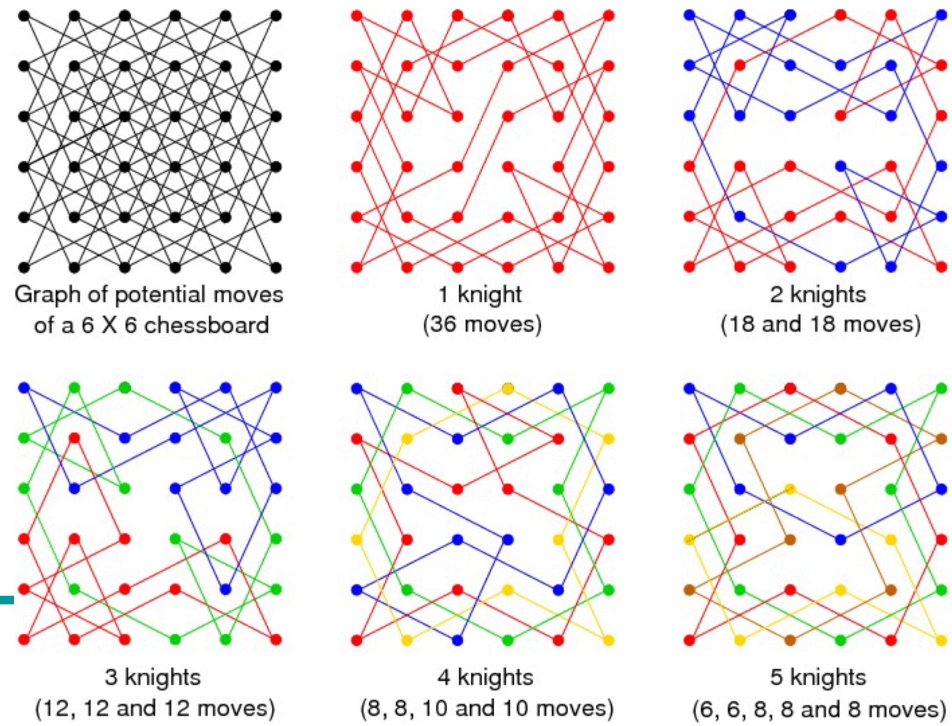
Contraintes globales

- **diffn**($[[O_{11},\ldots,O_{1n},L_{11},\ldots,L_{1n}],\ldots,[O_{m1},\ldots,O_{mn},L_{m1},\ldots,L_{mn}]]$) avec
  - m le nombre d'objets rectangulaires,
  - n la dimension des objets
  - O_{ij} position d'origine du i^{ème} objet dans la j^{ème} dimension
  - L_{ij} taille du i^{ème} objet dans la j^{ème} dimension

place sans recouvrement m objets rectangulaires i de dimension n de tailles $L_{i1},\ldots,L_{in}$

- **cycle**($N,[S_1,\ldots,S_m]$) avec
  - N le nombre de cycles
  - S_i, les cycles
  - m le nombre de nœuds du graphe

Génère le recouvrement d'un graphe par N cycles ou circuits. Chaque nœud n'appartient qu'à un cycle.



Applications

Remplissage de conteneurs (bin packing problem)

- Répartir n objets de taille L_i avec i=1,…,n dans au plus m conteneurs de capacité c.
- optimisation : Trouver le nombre minimal de conteneurs nécessaires.

Placement

- Placer sans recouvrement m objets rectangulaires i de dimension n de tailles $L_{i1},\ldots,L_{in}$

Tournée de véhicules (vehicule routing problem)

- Affecter chaque point de passage à un véhicule parmi n et préciser pour chaque véhicule l'ordre de visite.

Laurent.Bougrain@loria.fr

Systèmes de propagation de contraintes [Bockmayer, Hooker]

Système	Disponibilité	Contraintes	Langage	Site web
B-prolog	Commercial	Domaine fini	Prolog	http://www.picat-lang.org/bprolog/
CHIP	Commercial	Domaine fini, hybride, booléen, rationnel linéaire	C, C++	www.cosytec.com
Choco	Gratuit	Domaine fini	Claire	https://choco-solver.org
Eclipse	Gratuit si but non lucratif	Domaine fini, hybride	Prolog	eclipseclp.org/
GNU prolog	Gratuit	Domaine fini	Prolog	www.gprolog.org/
ILOG CPLEX	Commercial	Domaine fini, hybride	C++, Java, .net, python	https://www.ibm.com/fr-fr/products/ilog-cplex-optimization-studio
NCL	Commercial	domaine fini	Java	https://www.cs.cmu.edu/Groups/AI/lang/prolog/impl/parallel/ncl/0.html
Mozart	Gratuit	Domaine fini	Oz	mozart.github.io
Prolog IV	Commercial	Domaine fini, arithmétique linéaire et non linéaire par intervalles	Prolog	prolog-heritage.org/
Sicstus	Commercial	Domaine fini, booléen, réel/ rationnel linéaire	Prolog	sicstus.sics.se

Laurent.Bougrain@loria.fr