Cours de Master Recherche Spécialité CODE : Résolution de problèmes combinatoires

Christine Solnon

LIRIS, UMR 5205 CNRS / Université Lyon 1

2007

Rappel du plan du cours

16 heures de cours

- 1 - Introduction ($\sim$ 4 heures)
  - Qu'est-ce qu'un problème complexe ?
  - Exemples de problèmes complexes
- 2 - Approches complètes ($\sim$ 4 heures)
  - Structuration de l'espace de recherche en arbre
  - Techniques d'élagage et heuristiques d'ordre
- 3 - Approches incomplètes ($\sim$ 8 heures)
  - Approches basées sur le voisinage
  - Approches constructives

10 heures de TP + 4 heures d'exposés $\rightsquigarrow$ Carole Knibbe

Résolution du problème SAT par recherche de cliques

- Choix d'une méthode de résolution
  $\rightsquigarrow$ implémentation et expérimentation
- Rédaction d'un article et exposé oral

Plan de la deuxième partie : approches complètes

- Notions de correction et de complétude
- Résolution par Séparation & Evaluation (Branch & Bound)
  - Principe général
  - Application à la PLNE
  - Application à la planification
- Programmation par contraintes
  - Principe général
  - Propagation de contraintes et Consistances locales

Plan de la deuxième partie : approches complètes

Notions de correction et complétude

Correction et complétude d'un système formel

- Système formel = ensemble d'axiomes et règles d'inférences
- Un système formel S est
  - correct si toute assertion produite par S est vraie
  - complet si S peut engendrer toutes les assertions qui sont vraies

Correction et complétude d'un algorithme

Un algorithme A, calculant une réponse à un problème P, est

- correct si toute réponse calculée par A est solution de P
- complet si A peut calculer une réponse pour toute instance de P

Correction et complétude sont évidemment souhaitables...
...mais on les sacrifie parfois pour des raisons d'efficacités.

Notions de correction et complétude

Correction et complétude d'un système formel

- Système formel = ensemble d'axiomes et règles d'inférences
- Un système formel S est
  - correct si toute assertion produite par S est vraie
  - complet si S peut engendrer toutes les assertions qui sont vraies

Correction et complétude d'un algorithme

Un algorithme A, calculant une réponse à un problème P, est

- correct si toute réponse calculée par A est solution de P
- complet si A peut calculer une réponse pour toute instance de P

Correction et complétude sont évidemment souhaitables...
...mais on les sacrifie parfois pour des raisons d'efficacités.

Notions de correction et complétude

Correction et complétude d'un système formel

- Système formel = ensemble d'axiomes et règles d'inférences
- Un système formel S est
  - correct si toute assertion produite par S est vraie
  - complet si S peut engendrer toutes les assertions qui sont vraies

Correction et complétude d'un algorithme

Un algorithme A, calculant une réponse à un problème P, est

- correct si toute réponse calculée par A est solution de P
- complet si A peut calculer une réponse pour toute instance de P

Correction et complétude sont évidemment souhaitables...
...mais on les sacrifie parfois pour des raisons d'efficacités.

Résolution de problèmes NP-complets/difficiles (1)

1 - Exploration exhaustive de l'espace de recherche

- Structurer l'espace de recherche en arbre ou en treillis
- Filtrer et/ou borner $\rightsquigarrow$ couper des zones
- Heuristiques $\rightsquigarrow$ explorer en premier les zones prometteuses

$\rightsquigarrow$ Approches correctes et complètes... Complexité exponentielle

Deuxième partie du cours

2 - Exploration opportuniste de l'espace de recherche

- Approches basées sur le voisinage

  Recherche locale, Algorithmes génétiques
- Approches constructives

  Algorithmes gloutons, fourmis, EDA

$\rightsquigarrow$ Approches incomplètes... Complexité polynomiale

Troisième partie du cours

Résolution de problèmes NP-complets/difficiles (1)

1 - Exploration exhaustive de l'espace de recherche

- Structurer l'espace de recherche en arbre ou en treillis
- Filtrer et/ou borner $\rightsquigarrow$ couper des zones
- Heuristiques $\rightsquigarrow$ explorer en premier les zones prometteuses

$\rightsquigarrow$ Approches correctes et complètes... Complexité exponentielle

Deuxième partie du cours

2 - Exploration opportuniste de l'espace de recherche

- Approches basées sur le voisinage

  Recherche locale, Algorithmes génétiques
- Approches constructives

  Algorithmes gloutons, fourmis, EDA

$\rightsquigarrow$ Approches incomplètes... Complexité polynomiale

Troisième partie du cours

Résolution de problèmes NP-complets/difficiles (2)

3 - Résolution approximée

- Calcul d'une approximation de la solution optimale

  $\rightsquigarrow$ erreur bornée
- Temps de calcul dépend de l'erreur : erreur $\searrow$ temps $\nearrow$

$\rightsquigarrow$ Approches non correctes... Complexité polynomiale

Pas étudié dans ce cours

- Pour en savoir plus :

  http://legacy.orie.cornell.edu/$\sim$dpw/cornell.ps

Principe général des approches complètes

Approches top-down : construction d'un arbre de recherche

$\rightsquigarrow$ découper l'espace de recherche en portions de + en + petites

- racine $\rightsquigarrow$ espace de recherche initial
- noeud $\rightsquigarrow$ portion de l'espace de recherche
- 2 types de feuilles :
  - feuille "succès" $\rightsquigarrow$ solution du problème
  - feuille "échec" $\rightsquigarrow$ portion sans solution

$\rightsquigarrow$ Etudié dans ce cours

Approches bottom-up : construction d'un treillis

$\rightsquigarrow$ construire des « solutions » de plus en plus grandes

- niveau 1 du treillis $\rightsquigarrow$ solutions « de taille 1 »
- niveau k du treillis $\rightsquigarrow$ solutions « de taille k »
- haut du treillis $\rightsquigarrow$ solutions « de plus grande taille »

$\rightsquigarrow$ Pas étudié dans ce cours

Principe général des approches complètes

Approches top-down : construction d'un arbre de recherche

$\rightsquigarrow$ découper l'espace de recherche en portions de + en + petites

- racine $\rightsquigarrow$ espace de recherche initial
- noeud $\rightsquigarrow$ portion de l'espace de recherche
- 2 types de feuilles :
  - feuille "succès" $\rightsquigarrow$ solution du problème
  - feuille "échec" $\rightsquigarrow$ portion sans solution

$\rightsquigarrow$ Etudié dans ce cours

Approches bottom-up : construction d'un treillis

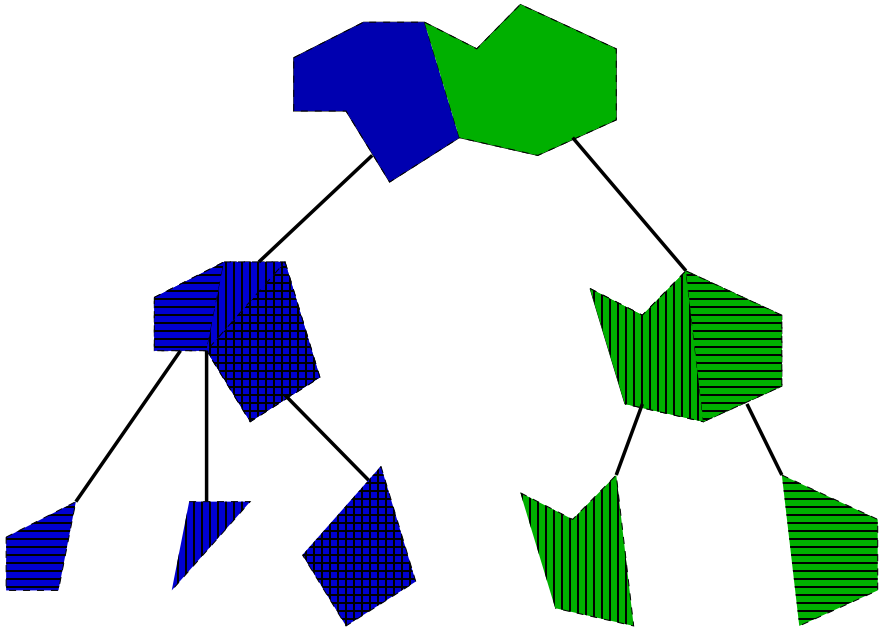
$\rightsquigarrow$ construire des $\ll$ solutions $\gg$ de plus en plus grandes

- niveau 1 du treillis $\rightsquigarrow$ solutions $\ll$ de taille 1 $\gg$
- niveau k du treillis $\rightsquigarrow$ solutions $\ll$ de taille k $\gg$
- haut du treillis $\rightsquigarrow$ solutions $\ll$ de plus grande taille $\gg$

$\rightsquigarrow$ Pas étudié dans ce cours

Plan de la deuxième partie : approches complètes

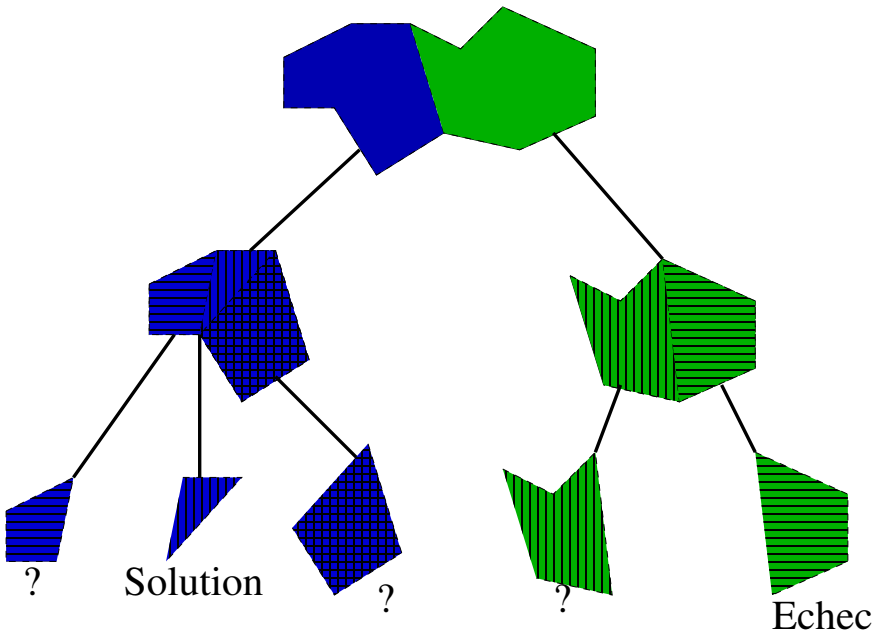
Construction d'un arbre de recherche



Définir des fonctions de :

- séparation $\rightsquigarrow$ partitionne un espace en sous-espaces
- évaluation $\rightsquigarrow$ solution, échec ou ?
  complexité (faiblement) polynomiale
- sélection $\rightsquigarrow$ prochain noeud à développer

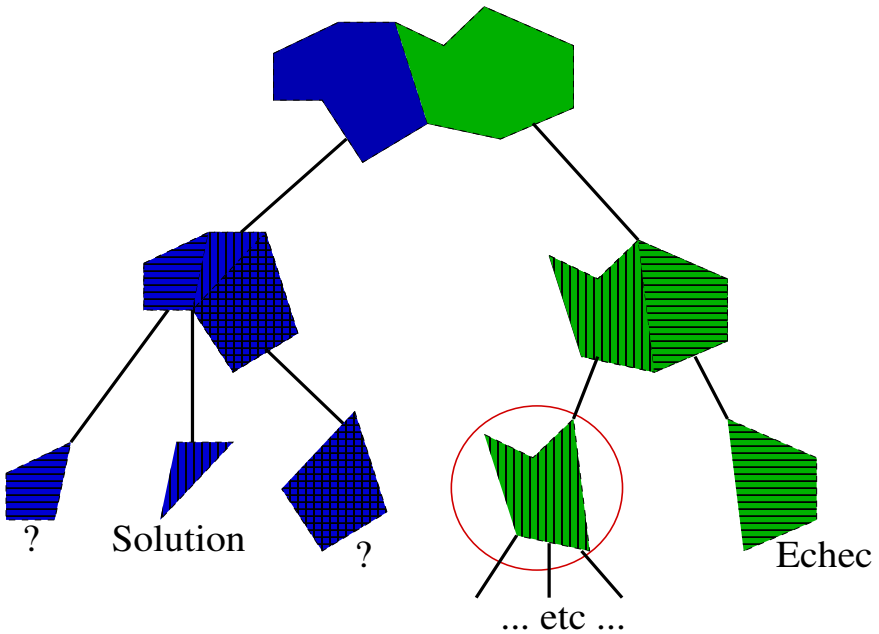
Construction d'un arbre de recherche



Définir des fonctions de :

- séparation $\rightsquigarrow$ partitionne un espace en sous-espaces
- évaluation $\rightsquigarrow$ solution, échec ou ?
  complexité (faiblement) polynomiale
- sélection $\rightsquigarrow$ prochain noeud à développer

Construction d'un arbre de recherche



Définir des fonctions de :

- séparation $\rightsquigarrow$ partitionne un espace en sous-espaces
- évaluation $\rightsquigarrow$ solution, échec ou ?
  complexité (faiblement) polynomiale
- sélection $\rightsquigarrow$ prochain noeud à développer

Exemple 1 : Programmation Linéaire en Nombres Entiers (1)

Exemple de PLNE

$$\begin{aligned} \text{maximiser} \quad & f = 3x + 4y - 2z \\ \text{tel que} \quad & 2x + z \geq 10 \\ & x - 3y \leq 30 \\ \text{et} \quad & x, y \text{ et } z \text{ sont des entiers} \end{aligned}$$

Complexité d'un PLNE

- Polynomiale si on enlève la contrainte "x, y et z sont des entiers"
  - $\rightsquigarrow$ Résolution par l'algorithme du simplex ou du point intérieur
- ... mais NP-difficile avec la contrainte "x, y et z sont des entiers"
  - $\rightsquigarrow$ Résolution par Séparation & évaluation

Exemple 1 : Programmation Linéaire en Nombres Entiers (1)

Exemple de PLNE

$$\begin{aligned} \text{maximiser} \quad & f = 3x + 4y - 2z \\ \text{tel que} \quad & 2x + z \geq 10 \\ & x - 3y \leq 30 \\ \text{et} \quad & x, y \text{ et } z \text{ sont des entiers} \end{aligned}$$

Complexité d'un PLNE

- Polynomiale si on enlève la contrainte "x, y et z sont des entiers"
  $\rightsquigarrow$ Résolution par l'algorithme du simplex ou du point intérieur
- ... mais NP-difficile avec la contrainte "x, y et z sont des entiers"
  $\rightsquigarrow$ Résolution par Séparation & évaluation

Exemple 1 : Programmation Linéaire en Nombres Entiers (2)

Résolution d'un PLNE par Séparation & Evaluation

- Fonction d'évaluation :

  Relâcher la contrainte "x, y et z sont des entiers"

  $\rightsquigarrow$ Calculer $x^*, y^*, z^* \in \mathbb{R}$ maximisant $f^* = 3x^* + 4y^* - 2z^*$

  - si x^*, y^* et z^* ont des valeurs entières, alors : solution
  - sinon si $f^* <$ meilleure solution trouvée alors : échec
  - sinon : continuer la recherche

- Fonction de sélection : choisir le nœud pour lequel f^* est maximal
- Fonction de séparation : couper le domaine d'une variable en 2

  (choisir une variable ayant une valeur non entière dans f^*)

Exemple 1 : Programmation Linéaire en Nombres Entiers (2)

Résolution d'un PLNE par Séparation & Evaluation

- Fonction d'évaluation :

  Relâcher la contrainte "x, y et z sont des entiers"

  $\rightsquigarrow$ Calculer $x^*, y^*, z^* \in \mathbb{R}$ maximisant $f^* = 3x^* + 4y^* - 2z^*$

  - si x^*, y^* et z^* ont des valeurs entières, alors : solution
  - sinon si $f^* <$ meilleure solution trouvée alors : échec
  - sinon : continuer la recherche

- Fonction de sélection : choisir le nœud pour lequel f^* est maximal
- Fonction de séparation : couper le domaine d'une variable en 2 (choisir une variable ayant une valeur non entière dans f^*)

Exemple 1 : Programmation Linéaire en Nombres Entiers (2)

Résolution d'un PLNE par Séparation & Evaluation

- Fonction d'évaluation :

  Relâcher la contrainte "x, y et z sont des entiers"

  $\rightsquigarrow$ Calculer $x^*, y^*, z^* \in \mathbb{R}$ maximisant $f^* = 3x^* + 4y^* - 2z^*$

  - si x^*, y^* et z^* ont des valeurs entières, alors : solution
  - sinon si $f^* <$ meilleure solution trouvée alors : échec
  - sinon : continuer la recherche

- Fonction de sélection : choisir le nœud pour lequel f^* est maximal
- Fonction de séparation : couper le domaine d'une variable en 2

  (choisir une variable ayant une valeur non entière dans f^*)

Exemple 1 : Programmation Linéaire en Nombres Entiers (2)

Résolution d'un PLNE par Séparation & Evaluation

- Fonction d'évaluation :

  Relâcher la contrainte "x, y et z sont des entiers"

  $\rightsquigarrow$ Calculer $x^*, y^*, z^* \in \mathbb{R}$ maximisant $f^* = 3x^* + 4y^* - 2z^*$

  - si x^*, y^* et z^* ont des valeurs entières, alors : solution
  - sinon si $f^* <$ meilleure solution trouvée alors : échec
  - sinon : continuer la recherche

- Fonction de sélection : choisir le nœud pour lequel f^* est maximal
- Fonction de séparation : couper le domaine d'une variable en 2

  (choisir une variable ayant une valeur non entière dans f^*)

Pour en savoir plus

http://mat.gsia.cmu.edu/orclass/integer/integer.html

Exemple 2 : Problèmes de planification (1)

Problèmes de planification (rappel)

- Définis par
  - Un ensemble (éventuellement infini) d'états E
  - Un état initial $E_0 \in E$
  - Un ensemble d'états finaux $F \subseteq E$
  - Un ensemble d'op. O permettant de passer d'états en états $E_i \xrightarrow{O_k} E_j$ si O_k permet de passer de E_i à E_j
- Un plan est une suite d'opérations permettant de passer de l'état initial E_0 à un état final de F

$$\text{Plan} = E_0 \xrightarrow{O_1} E_1 \xrightarrow{O_2} E_2 \ldots E_{n-1} \xrightarrow{O_n} E_n \text{ avec } E_n \in F$$

- Problème = trouver le plan de coût minimal
- Espace de recherche = ensemble des suites d'opérations partant de E_0

Exemple 2 : Problèmes de planification (2)

Recherche du plan de coût minimal (algorithme A*)

- Construction d'un arbre de recherche
  - Nœud de l'arbre $\rightsquigarrow$ suite d'opérations partant de E_0
  - Racine $\rightsquigarrow$ suite vide d'opérations
- Fonction de séparation $\rightsquigarrow$ 1 fils pour chaque opération possible
- Fonction d'évaluation : soient
  - $g(n)$ = coût des opérations faites de la racine jusqu'à n
  - $h(n)$ = borne inférieure du coût des opérations restant à faire pour aller de n jusqu'à un état final

  Fonction d'évaluation : $f(n) = g(n) + h(n)$

  $\rightsquigarrow$ Estimation du coût du meilleur plan passant par n

  $\rightsquigarrow$ $f(n) >$ coût du meilleur plan trouvé $\Rightarrow$ couper le nœud n
- Fonction de sélection $\rightsquigarrow$ noeud ayant la meilleure évaluation $f(n)$

Exemple 2 : Problèmes de planification (2)

Recherche du plan de coût minimal (algorithme A*)

- Construction d'un arbre de recherche
  - Nœud de l'arbre $\rightsquigarrow$ suite d'opérations partant de E_0
  - Racine $\rightsquigarrow$ suite vide d'opérations
- Fonction de séparation $\rightsquigarrow$ 1 fils pour chaque opération possible
- Fonction d'évaluation : soient
  - $g(n)$ = coût des opérations faites de la racine jusqu'à n
  - $h(n)$ = borne inférieure du coût des opérations restant à faire pour aller de n jusqu'à un état final

  Fonction d'évaluation : $f(n) = g(n) + h(n)$

  $\rightsquigarrow$ Estimation du coût du meilleur plan passant par n

  $\rightsquigarrow$ $f(n) >$ coût du meilleur plan trouvé $\Rightarrow$ couper le nœud n
- Fonction de sélection $\rightsquigarrow$ noeud ayant la meilleure évaluation $f(n)$

Exemple 2 : Problèmes de planification (2)

Recherche du plan de coût minimal (algorithme A*)

- Construction d'un arbre de recherche
  - Nœud de l'arbre $\rightsquigarrow$ suite d'opérations partant de E_0
  - Racine $\rightsquigarrow$ suite vide d'opérations
- Fonction de séparation $\rightsquigarrow$ 1 fils pour chaque opération possible
- Fonction d'évaluation : soient
  - $g(n)$ = coût des opérations faites de la racine jusqu'à n
  - $h(n)$ = borne inférieure du coût des opérations restant à faire pour aller de n jusqu'à un état final

  Fonction d'évaluation : $f(n) = g(n) + h(n)$

  $\rightsquigarrow$ Estimation du coût du meilleur plan passant par n

  $\rightsquigarrow$ $f(n) >$ coût du meilleur plan trouvé $\Rightarrow$ couper le nœud n
- Fonction de sélection $\rightsquigarrow$ noeud ayant la meilleure évaluation $f(n)$

Exemple 2 : Problèmes de planification (2)

Recherche du plan de coût minimal (algorithme A*)

- Construction d'un arbre de recherche
  - Nœud de l'arbre $\rightsquigarrow$ suite d'opérations partant de E_0
  - Racine $\rightsquigarrow$ suite vide d'opérations
- Fonction de séparation $\rightsquigarrow$ 1 fils pour chaque opération possible
- Fonction d'évaluation : soient
  - $g(n)$ = coût des opérations faites de la racine jusqu'à n
  - $h(n)$ = borne inférieure du coût des opérations restant à faire pour aller de n jusqu'à un état final

  Fonction d'évaluation : $f(n) = g(n) + h(n)$

  $\rightsquigarrow$ Estimation du coût du meilleur plan passant par n

  $\rightsquigarrow$ $f(n) >$ coût du meilleur plan trouvé $\Rightarrow$ couper le nœud n
- Fonction de sélection $\rightsquigarrow$ noeud ayant la meilleure évaluation $f(n)$

Exemple 2 : Problèmes de planification (3)

- Exemple de problème de planification : le ≪ taquin ≫
  - Etat initial =

	B	C
A	D	F
G	E	H

  - Etat final =

A	B	C
D	E	F
G	H	

  - Opération = échanger la case vide avec une case adj.
  - But = trouver le plus petit plan (en nombre d'opérations)
- Coût du début du plan = nombre d'opérations déjà effectuées
- Estimation du coût pour atteindre l'état final = ???

B	D	C
	A	F
G	E	H

Exemple 2 : Problèmes de planification (3)

- Exemple de problème de planification : le ≪ taquin ≫
  - Etat initial =

	B	C
A	D	F
G	E	H

  - Etat final =

A	B	C
D	E	F
G	H	

  - Opération = échanger la case vide avec une case adj.
  - But = trouver le plus petit plan (en nombre d'opérations)
- Coût du début du plan = nombre d'opérations déjà effectuées
- Estimation du coût pour atteindre l'état final = dist de Manhattan

B	D	C
	A	F
G	E	H

$\delta(A) = 2 \quad \delta(D) = 2 \quad \delta(G) = 0$
$\delta(B) = 1 \quad \delta(E) = 1 \quad \delta(H) = 1$
$\delta(C) = 0 \quad \delta(F) = 0$

$\Rightarrow$ au moins 7 opérations pour aller jusqu'à l'état final

Plan de la deuxième partie : approches complètes

”Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming:

the user states the problem,
the computer solves it.”

Eugene C. Freuder

La programmation par Contraintes (PPC)

- Spécifier le problème en termes de contraintes

  $\rightsquigarrow$ Problèmes de satisfaction de contraintes (CSPs)

- Concevoir des algorithmes de résolution de CSPs

  $\rightsquigarrow$ Solveurs de contraintes

- Intégrer ces solveurs dans un langage

  $\rightsquigarrow$ Le programmeur définit les contraintes...
  ... et le solveur cherche la solution.

  Exemples de langages de PPC :

  - ALICE [Jean-Louis Laurière, 1976]
  - CHIP, Prolog V, Gnu-Prolog
  - CHOCO, Ilog solver, Gecode
  - ...

- Pas toujours aussi efficace qu'un programme « cousu main »

  ... mais tellement plus vite développé !

Programme des reines avec Ilog Solver

```
#include <ilsolver/ilcint.h>
void main(){
    int nbReines = 100;
    IlcManager m(); // m est le solveur de contraintes
    // On déclare les variables et leur domaine
    IlcIntVarArray X(m,nbReines,0,nbReines-1);
    // On pose les contraintes
    for (int i=0; i<n; i++)
        for (int j=i+1; j<n; j++){
            m.add(X[i]+i != X[j]+j);
            m.add(X[i]-i != X[j]-j);
        }
    m.add(IlcAllDiff(X));
    // On demande au solveur de chercher une solution
    m.nextSolution();
    for (int i=0; i<n; i++)
        cout << "Reine : col. " << i << " ligne " << X[i].getValue();
}
```

Problèmes de Satisfaction de Contraintes (1)

Définition d'un CSP (rappel)

CSP = (X, D, C) tel que

- $X = \{X_1, X_2, \ldots, X_n\}$ est l'ensemble des variables
- $D(X_i)$ est le domaine de X_i
- C est l'ensemble des contraintes

Solution d'un CSP (rappel)

- Affectation = ensemble de couples $< X_i, v_i >$ tq $v_i \in D(X_i)$
- Affectation partielle $\rightsquigarrow$ certaines variables ne sont pas affectées
- Affectation totale $\rightsquigarrow$ toutes les variables sont affectées
- Affectation consistante $\rightsquigarrow$ contraintes satisfaites

Solution = affectation complète consistante

Problèmes de Satisfaction de Contraintes (1)

Définition d'un CSP (rappel)

CSP = (X, D, C) tel que

- $X = \{X_1, X_2, \ldots, X_n\}$ est l'ensemble des variables
- $D(X_i)$ est le domaine de X_i
- C est l'ensemble des contraintes

Solution d'un CSP (rappel)

- Affectation = ensemble de couples $< X_i, v_i >$ tq $v_i \in D(X_i)$
- Affectation partielle $\rightsquigarrow$ certaines variables ne sont pas affectées
- Affectation totale $\rightsquigarrow$ toutes les variables sont affectées
- Affectation consistante $\rightsquigarrow$ contraintes satisfaites

Solution = affectation complète consistante

Problèmes de Satisfaction de Contraintes (2)

"Résoudre" un CSP (X, D, C) peut signifier :

- trouver une solution,
- prouver qu'il existe au moins une solution (consistance),
- trouver les intervalles de valeurs dans lesquels se trouvent les solutions,
- trouver toutes les solutions (si énumérable),
- trouver la "meilleure" solution...
  ... par rapport à une fonction objectif donnée
- trouver une affectation qui maximise le nb de contraintes satisfaites (MAX-CSPs, CSPs valués, Contraintes "soft", ...)

Techniques de résolution de CSPs

Double origine

Intelligence Artificielle et Recherche Opérationnelle

Dépend du type des domaines et contraintes

- Domaines continus (réels) / contraintes numériques

  Algorithme de Gauss, Simplex, bases de Grobner...
  Arithmétique des intervalles

- Domaines discrets (énumérables $\longrightarrow$ entiers)

  $\rightsquigarrow$ résolution par Séparation & Propagation

- Domaines symboliques (chaînes, graphes, ...)

  $\rightsquigarrow$ algorithmes “dédiés”

Résolution par Séparation & Propagation (1)

Construction d'un arbre de recherche

- Racine = affectation vide
- Noeuds internes = affectations partielles consistantes
- 2 types de feuilles =
  - Affectations partielles inconsistantes $\rightsquigarrow$ échecs
  - Affectations totales consistantes $\rightsquigarrow$ solutions

Fonction de séparation

$\rightsquigarrow$ Séparer le domaine courant d'une variable en plusieurs parties

- choisir une variable dont le domaine contient plusieurs valeurs
- découper le domaine en sous-domaines et créer autant de fils que de sous-domaines

$\rightsquigarrow$ Heuristique de choix de la variable

Résolution par Séparation & Propagation (1)

Construction d'un arbre de recherche

- Racine = affectation vide
- Noeuds internes = affectations partielles consistantes
- 2 types de feuilles =
  - Affectations partielles inconsistantes $\rightsquigarrow$ échecs
  - Affectations totales consistantes $\rightsquigarrow$ solutions

Fonction de séparation

$\rightsquigarrow$ Séparer le domaine courant d'une variable en plusieurs parties

- choisir une variable dont le domaine contient plusieurs valeurs
- découper le domaine en sous-domaines et créer autant de fils que de sous-domaines

$\rightsquigarrow$ Heuristique de choix de la variable

Résolution par Séparation & Propagation (2)

Fonction de sélection

$\rightsquigarrow$ Choisir le prochain noeud à développer

- en général : en profondeur d'abord
  $\rightsquigarrow$ moins coûteux en mémoire
- Heuristique sur l'ordre de développement des fils
  $\rightsquigarrow$ choisir en premier les valeurs les plus $\ll$ prometteuses $\gg$

Fonction d'évaluation

$\rightsquigarrow$ propagation des contraintes

- Vérifier que les variables affectées satisfont toutes les contraintes
  $\rightsquigarrow$ $\ll$ retour-arrière simple $\gg$ (simple backtrack)
- Filtrer les domaines des variables non affectées
  $\rightsquigarrow$ vérification d'une consistance locale

Résolution par Séparation & Propagation (2)

Fonction de sélection

$\rightsquigarrow$ Choisir le prochain noeud à développer

- en général : en profondeur d'abord
  $\rightsquigarrow$ moins coûteux en mémoire
- Heuristique sur l'ordre de développement des fils
  $\rightsquigarrow$ choisir en premier les valeurs les plus $\ll$ prometteuses $\gg$

Fonction d'évaluation

$\rightsquigarrow$ propagation des contraintes

- Vérifier que les variables affectées satisfont toutes les contraintes
  $\rightsquigarrow$ $\ll$ retour-arrière simple $\gg$ (simple backtrack)
- Filtrer les domaines des variables non affectées
  $\rightsquigarrow$ vérification d'une consistance locale

Résolution par Séparation & Propagation (3)

Algorithme

$A \leftarrow \emptyset$
Tant qu'il reste des variables non affectées dans A
$\rightsquigarrow$ choisir une variable X_i non affectée dans A
$\rightsquigarrow$ choisir une valeur $v \in D(X_i)$ pour cette variable
$\rightsquigarrow$ si propagation($X_i = v, C, D$)=échec alors "backtrack"

Propagation de contraintes et Consistances locales

$\rightsquigarrow$ Différents niveaux de consistance locale

- Consistance de noeud :

  $\forall X_i, \forall v \in D(X_i), \{< X_i, v >\} \cup A$ est consistant
- Consistance d'arc :

  $\forall X_i, \forall v \in D(X_i), \forall X_j, \exists u \in D(X_j)$:
  $\{< X_i, v >, < X_j, u >\} \cup A$ est consistant
- ...

Contraintes globales et algorithmes dédiés

Généricité vs Efficacité

Les solveurs de contraintes peuvent résoudre n'importe quel CSP. En contrepartie, ils sont parfois moins efficaces que des approches dédiées

Notion de contrainte globale

Introduire de nouvelles contraintes pour certains problèmes "typiques" et intégrer au solveur de contraintes des algorithmes dédiés à ces problèmes

Exemples de contraintes globales

- allDiff(L)
- atmost(L,V,N)
- path(G, V_1, V_2)
- graphIsomorphism(G_1, G_2)
- ...