

**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

- 1 - Introduction
 - Qu'est-ce qu'un problème complexe ?
 - Exemples de problèmes complexes
- 2 - Approches complètes
 - Structuration de l'espace de recherche en Arbre
 - Application à la planification et la résolution de CSPs
 - Structuration de l'espace de recherche en Treillis
 - 6/11/2007 et 8/11/2007 : **Fabien De Marchi**
- 3 - Approches incomplètes
 - Algorithmes gloutons
 - Recherche locale
 - Algorithmes génétiques
 - 30/10/2007 : **Guillaume Beslon**
- 4 - Optimisation par colonies de fourmis
 - Principes généraux de la méta-heuristique ACO
 - Applications
 - Diversification vs intensification de la recherche

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
 - Contraintes globales et algorithmes de filtrage associés
↪ illustration sur le problème de l'isomorphisme de graphes
 - Résolution par construction d'un treillis
- 6/11/2007 et 8/11/2007 : Fabien De Marchi

Plan de la deuxième partie : approches complètes

- 1 Correction et complétude
- 2 Séparation & Evaluation
- 3 Programmation par contraintes

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 les instances
Recherche locale, Algorithmes génétiques
- Approches basées sur les modèles
Algorithmes gloutons, fourmis, EDA

\rightsquigarrow Approches incomplètes... Complexité polynomiale

Troisième et quatrième parties 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 les instances
Recherche locale, Algorithmes génétiques
- Approches basées sur les modèles
Algorithmes gloutons, fourmis, EDA

\rightsquigarrow Approches incomplètes... Complexité polynomiale

Troisième et quatrième parties 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/dpw/cornell.ps>

Principe général des approches complètes

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

↪ découper l'espace de recherche en portions de + en + petites

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

↪ Étudié aux points 2.2 et 2.3 du cours

Approches bottom-up : construction d'un treillis

↪ construire des < solutions > de plus en plus grandes

- niveau 1 du treillis ↪ solutions < de taille 1 >
- niveau k du treillis ↪ solutions < de taille k >
- haut du treillis ↪ solutions < de plus grande taille >

↪ Étudié au point 2.4 du cours avec Fabien De Marchi

Principe général des approches complètes

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

↪ découper l'espace de recherche en portions de + en + petites

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

↪ Étudié aux points 2.2 et 2.3 du cours

Approches bottom-up : construction d'un treillis

↪ construire des « solutions » de plus en plus grandes

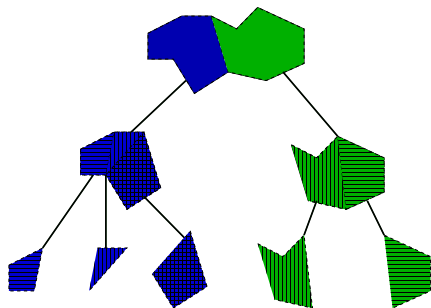
- niveau 1 du treillis ↪ solutions « de taille 1 »
- niveau k du treillis ↪ solutions « de taille k »
- haut du treillis ↪ solutions « de plus grande taille »

↪ Étudié au point 2.4 du cours avec Fabien De Marchi

Plan de la deuxième partie : approches complètes

- 1 Correction et complétude
- 2 Séparation & Evaluation**
- 3 Programmation par contraintes

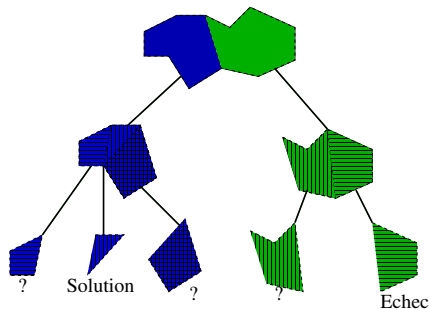
Construction d'un arbre de recherche



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

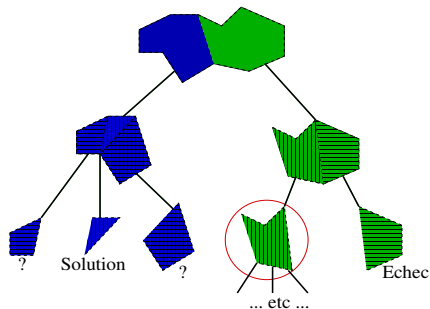
Construction d'un arbre de recherche



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

maximiser $f = 3x + 4y - 2z$
tel que $2x + z \geq 10$
 $x - 3y \leq 30$
et x, y et z sont des entiers

Complexité d'un PLNE

- Polynomiale si on enlève la contrainte “ x, y et z sont des entiers”
↪ Résolution par l'algorithme du simplexe ou du point intérieur
- ... mais NP-difficile avec la contrainte “ x, y et z sont des entiers”
↪ Résolution par Séparation & évaluation

Exemple 1 : Programmation Linéaire en Nombres Entiers (1)

Exemple de PLNE

maximiser $f = 3x + 4y - 2z$
tel que $2x + z \geq 10$
 $x - 3y \leq 30$
et x, y et z sont des entiers

Complexité d'un PLNE

- Polynomiale si on enlève la contrainte “ x, y et z sont des entiers”
↪ Résolution par l'algorithme du simplexe ou du point intérieur
- ... mais NP-difficile avec la contrainte “ x, y et z sont des entiers”
↪ 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"

↪ 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"

↪ 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 & Évaluation

- Fonction d'évaluation :

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

↪ 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”

↪ 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

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

- 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 nœud 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 nœud 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

$\rightsquigarrow f(n) > \text{coût du meilleur plan trouvé} \Rightarrow$ couper le nœud n

- Fonction de sélection \rightsquigarrow nœud 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 nœud 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

$$\delta(A) = 2 \quad \delta(D) = 2 \quad \delta(G) = 0$$

$$\delta(B) = 1 \quad \delta(E) = 1 \quad \delta(H) = 1$$

$$\delta(C) = 0 \quad \delta(F) = 0$$

⇒ au moins 7 opérations pour aller jusqu'à l'état final

Plan de la deuxième partie : approches complètes

- 1 Correction et complétude
- 2 Séparation & Evaluation
- 3 Programmation par contraintes**

"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
 - ↪ Problèmes de satisfaction de contraintes (CSPs)
- Concevoir des algorithmes de résolution de CSPs
 - ↪ Solveurs de contraintes
- Intégrer ces solveurs dans un langage
 - ↪ 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

- $X = \{X_1, X_2, \dots, X_n\}$ 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 $\langle X_i, v_i \rangle$ 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

- $X = \{X_1, X_2, \dots, X_n\}$ 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 $\langle X_i, v_i \rangle$ 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 \rightarrow 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

↪ Choisir le prochain noeud à développer

- en général : en profondeur d'abord
 - ↪ moins coûteux en mémoire
- Heuristique sur l'ordre de développement des fils
 - ↪ choisir en premier les valeurs les plus « prometteuses »

Fonction d'évaluation

↪ propagation des contraintes

- Vérifier que les variables affectées satisfont toutes les contraintes
 - ↪ « retour-arrière simple » (simple backtrack)
- Filtrer les domaines des variables non affectées
 - ↪ vérification d'une consistance locale

Résolution par Séparation & Propagation (2)

Fonction de sélection

↪ Choisir le prochain noeud à développer

- en général : en profondeur d'abord
 - ↪ moins coûteux en mémoire
- Heuristique sur l'ordre de développement des fils
 - ↪ choisir en premier les valeurs les plus « prometteuses »

Fonction d'évaluation

↪ propagation des contraintes

- Vérifier que les variables affectées satisfont toutes les contraintes
 - ↪ « retour-arrière simple » (simple backtrack)
- Filtrer les domaines des variables non affectées
 - ↪ vérification d'une consistance locale

Résolution par Séparation & Propagation (3)

Algorithme

$A \leftarrow \emptyset$

Tant qu'il reste des variables non affectées dans A

↪ choisir une variable X_i non affectée dans A

↪ choisir une valeur $v \in D(X_i)$ pour cette variable

↪ si $\text{propagation}(X_i = v, C, D) = \text{échec}$ alors "backtrack"

Propagation de contraintes et Consistances locales

↪ Différents niveaux de consistance locale

- Consistance de noeud :

$\forall X_i, \forall v \in D(X_i), \{ \langle X_i, v \rangle \} \cup A$ est consistant

- Consistance d'arc :

$\forall X_i, \forall v \in D(X_i), \forall X_j, \exists u \in D(X_j) :$

$\{ \langle X_i, v \rangle, \langle X_j, u \rangle \} \cup A$ 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

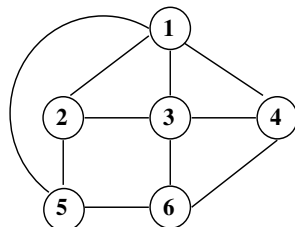
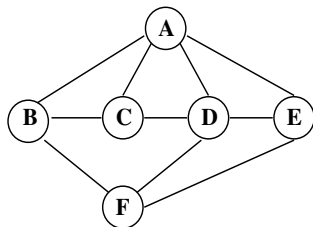
- allDiff(L)
- atmost(L, V, N)
- path(G, V_1, V_2)
- graphIsomorphism(G_1, G_2)

Example: Graph Isomorphism Problem (GIP)

Definition

Two Graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic iff there exists a bijection $f : V \rightarrow V'$ such that

$$\forall (v_1, v_2) \in V^2, (v_1, v_2) \in E \Leftrightarrow (f(v_1), f(v_2)) \in E'$$

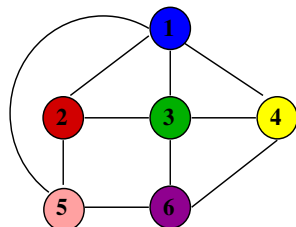
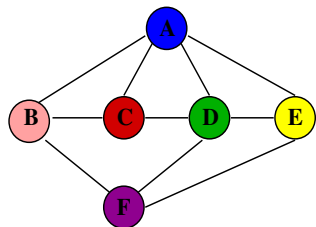


Example: Graph Isomorphism Problem (GIP)

Definition

Two Graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic iff there exists a bijection $f : V \rightarrow V'$ such that

$$\forall (v_1, v_2) \in V^2, (v_1, v_2) \in E \Leftrightarrow (f(v_1), f(v_2)) \in E'$$



$$f(A) = 1, f(B) = 5, f(C) = 2, f(D) = 3, f(E) = 4, f(F) = 6$$

Example: Graph Isomorphism Problem (GIP)

Complexity of the GIP

Still an open question ... between \mathcal{P} and \mathcal{NP} -complete ?

Approaches for solving GIPs

- Dedicated algorithms (Nauty [McKay 81])
 - Very efficient \rightsquigarrow tractable on large graphs
 - Not flexible \rightsquigarrow new constraints can't be added
- Constraint Programming
 - Flexible \rightsquigarrow new constraints can easily be added
 - Not very efficient \rightsquigarrow limited to rather small graphs

Solving GIP with CP

CSP associated with a GIP

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs

- Associate a variable X_i with each vertex $i \in V_1$,
- Domain of X_i = vertices of V_2 having the same degree than i
- Constraints: $\forall (i, j) \in V_1^2, (i, j) \in E_1 \Leftrightarrow (X_i, X_j) \in E_2$

Problem: global semantics decomposed into binary constraints

How to make CP competitive with dedicated approaches ?

- Introduce a global constraint for the GIP

$$gip(V_1, E_1, V_2, E_2, L)$$

(where L associates a different variable to each vertex of V_1)

- Define a filtering algorithm for this global constraint
 \rightsquigarrow use a labelling function to filter domains

Solving GIP with CP

CSP associated with a GIP

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs

- Associate a variable X_i with each vertex $i \in V_1$,
- Domain of X_i = vertices of V_2 having the same degree than i
- Constraints: $\forall (i, j) \in V_1^2, (i, j) \in E_1 \Leftrightarrow (X_i, X_j) \in E_2$

Problem: global semantics decomposed into binary constraints

How to make CP competitive with dedicated approaches ?

- Introduce a global constraint for the GIP

$$gip(V_1, E_1, V_2, E_2, L)$$

(where L associates a different variable to each vertex of V_1)

- Define a filtering algorithm for this global constraint
 \rightsquigarrow use a labelling function to filter domains

Filtering based on labelling

Definition (labelling function α)

Given a graph $G = (V, E)$ and a vertex $v \in V$,

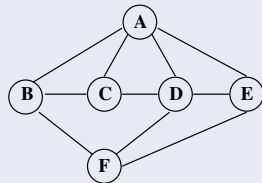
$\alpha_G(v) \rightsquigarrow$ label associated with v by the labelling function α

Definition (isomorphic-consistent labelling function)

For every pair of isomorphic graphs $G = (V, E)$ and $G' = (V', E')$, for every isomorphism function $f : V \rightarrow V'$ and every vertex $v \in V$,

$$\alpha_G(v) = \alpha_{G'}(f(v))$$

Example: labelling function based on degrees



$$\alpha_G(A) = \alpha_G(D) = 4$$

$$\alpha_G(B) = \alpha_G(C) = \alpha_G(E) = \alpha_G(F) = 3$$

Strengthening a labelling

Definition (re-labelling)

Given a graph $G = (V, E)$ and a labelling function α_G^i ,

$$\alpha_G^{i+1}(v) = \alpha_G^i(v) \cdot \{(k, l), \quad l \text{ is a label and} \\ k = \text{nb of neighbours of } v \text{ labelled by } l\}$$

$$\alpha_G^0 : A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1 : \begin{array}{l} A, B, D \quad \rightarrow \emptyset \cdot \{(4, \emptyset)\} \\ C, E, F, G \quad \rightarrow \emptyset \cdot \{(3, \emptyset)\} \end{array}$$

$$\begin{array}{l} A \quad \rightarrow \circ \cdot \{(2, \circ), (2, \cdot)\} \\ E, F \quad \rightarrow \cdot \cdot \{(2, \circ), (1, \cdot)\} \end{array}$$

$$\alpha_G^2 : \begin{array}{l} B, D \quad \rightarrow \circ \cdot \{(1, \circ), (3, \cdot)\} \\ C \quad \rightarrow \cdot \cdot \{(3, \circ)\} \\ G \quad \rightarrow \cdot \cdot \{(1, \circ), (2, \cdot)\} \end{array}$$

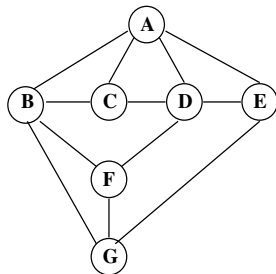
$$\alpha_G^3 : \dots \text{ etc } \dots$$

Strengthening a labelling

Definition (re-labelling)

Given a graph $G = (V, E)$ and a labelling function α_G^i ,

$$\alpha_G^{i+1}(v) = \alpha_G^i(v) \cdot \{(k, l), \quad l \text{ is a label and} \\ k = \text{nb of neighbours of } v \text{ labelled by } l\}$$



$$\alpha_G^0 : A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1 : \begin{array}{l} A, B, D \rightarrow \emptyset \cdot \{(4, \emptyset)\} \\ C, E, F, G \rightarrow \emptyset \cdot \{(3, \emptyset)\} \end{array}$$

$$\alpha_G^2 : \begin{array}{l} A \rightarrow \emptyset \cdot \{(2, \circ), (2, _)\} \\ E, F \rightarrow _ \cdot \{(2, \circ), (1, _)\} \end{array}$$

$$\alpha_G^3 : \begin{array}{l} B, D \rightarrow \emptyset \cdot \{(1, \circ), (3, _)\} \\ C \rightarrow _ \cdot \{(3, \circ)\} \\ G \rightarrow _ \cdot \{(1, \circ), (2, _)\} \end{array}$$

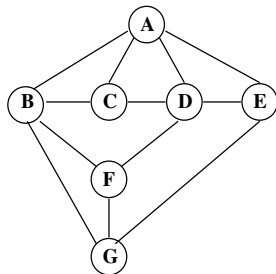
$$\alpha_G^4 : \dots \text{ etc } \dots$$

Strengthening a labelling

Definition (re-labelling)

Given a graph $G = (V, E)$ and a labelling function α_G^i ,

$$\alpha_G^{i+1}(v) = \alpha_G^i(v) \cdot \{(k, l), \quad l \text{ is a label and} \\ k = \text{nb of neighbours of } v \text{ labelled by } l\}$$



$$\alpha_G^0 : A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1 : \begin{array}{ll} A, B, D & \rightarrow \emptyset \cdot \{(4, \emptyset)\} \\ C, E, F, G & \rightarrow \emptyset \cdot \{(3, \emptyset)\} \end{array}$$

$$\alpha_G^2 : \begin{array}{ll} A & \rightarrow \circ \cdot \{(2, \circ), (2, _)\} \\ E, F & \rightarrow \cdot \{(2, \circ), (1, _)\} \\ B, D & \rightarrow \circ \cdot \{(1, \circ), (3, _)\} \\ C & \rightarrow \cdot \{(3, \circ)\} \\ G & \rightarrow \cdot \{(1, \circ), (2, _)\} \end{array}$$

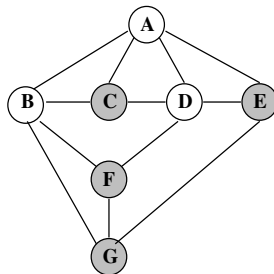
$$\alpha_G^3 : \dots \text{ etc } \dots$$

Strengthening a labelling

Definition (re-labelling)

Given a graph $G = (V, E)$ and a labelling function α_G^i ,

$$\alpha_G^{i+1}(v) = \alpha_G^i(v) \cdot \{(k, l), \quad l \text{ is a label and} \\ k = \text{nb of neighbours of } v \text{ labelled by } l\}$$



$$\alpha_G^0 : A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1 : \begin{array}{ll} A, B, D & \rightarrow \emptyset.\{(4, \emptyset)\} \Rightarrow \bigcirc \\ C, E, F, G & \rightarrow \emptyset.\{(3, \emptyset)\} \Rightarrow \bullet \end{array}$$

$$A \rightarrow \bigcirc.\{(2, \bigcirc), (2, _)\}$$

$$E, F \rightarrow _.\{(2, \bigcirc), (1, _)\}$$

$$\alpha_G^2 : B, D \rightarrow \bigcirc.\{(1, \bigcirc), (3, _)\}$$

$$C \rightarrow _.\{(3, \bigcirc)\}$$

$$G \rightarrow _.\{(1, \bigcirc), (2, _)\}$$

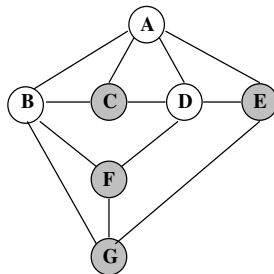
$$\alpha_G^3 : \dots \text{ etc } \dots$$

Strengthening a labelling

Definition (re-labelling)

Given a graph $G = (V, E)$ and a labelling function α_G^i ,

$$\alpha_G^{i+1}(v) = \alpha_G^i(v) \cdot \{(k, l) \mid l \text{ is a label and } k = \text{nb of neighbours of } v \text{ labelled by } l\}$$



$$\alpha_G^0: A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1: \begin{array}{l} A, B, D \rightarrow \emptyset \cdot \{(4, \emptyset)\} \Rightarrow \bigcirc \\ C, E, F, G \rightarrow \emptyset \cdot \{(3, \emptyset)\} \Rightarrow \bullet \end{array}$$

$$A \rightarrow \bigcirc \cdot \{(2, \bigcirc), (2, \bullet)\}$$

$$E, F \rightarrow \bullet \cdot \{(2, \bigcirc), (1, \bullet)\}$$

$$\alpha_G^2: B, D \rightarrow \bigcirc \cdot \{(1, \bigcirc), (3, \bullet)\}$$

$$C \rightarrow \bullet \cdot \{(3, \bigcirc)\}$$

$$G \rightarrow \bullet \cdot \{(1, \bigcirc), (2, \bullet)\}$$

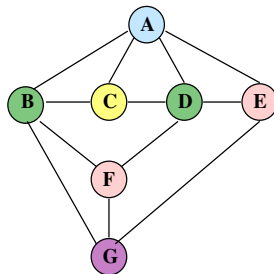
$$\alpha_G^3: \dots \text{ etc } \dots$$

Strengthening a labelling

Definition (re-labelling)

Given a graph $G = (V, E)$ and a labelling function α_G^i ,

$$\alpha_G^{i+1}(v) = \alpha_G^i(v) \cdot \{(k, l), \quad l \text{ is a label and} \\ k = \text{nb of neighbours of } v \text{ labelled by } l\}$$



$$\alpha_G^0 : A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1 : \begin{array}{l} A, B, D \rightarrow \emptyset \cdot \{(4, \emptyset)\} \Rightarrow \circ \\ C, E, F, G \rightarrow \emptyset \cdot \{(3, \emptyset)\} \Rightarrow \bullet \end{array}$$

$$\begin{array}{l} A \rightarrow \circ \cdot \{(2, \circ), (2, \bullet)\} \Rightarrow \text{light blue circle} \\ E, F \rightarrow \bullet \cdot \{(2, \circ), (1, \bullet)\} \Rightarrow \text{pink circle} \end{array}$$

$$\alpha_G^2 : \begin{array}{l} B, D \rightarrow \circ \cdot \{(1, \circ), (3, \bullet)\} \Rightarrow \text{green circle} \\ C \rightarrow \bullet \cdot \{(3, \circ)\} \Rightarrow \text{yellow circle} \\ G \rightarrow \bullet \cdot \{(1, \circ), (2, \bullet)\} \Rightarrow \text{purple circle} \end{array}$$

$$\alpha_G^3 : \dots \text{ etc } \dots$$

Properties of re-labelling

Theorem 1: Correctness

Given an isomorphic-consistent labelling function α^i , the labelling function α^{i+1} is also isomorphic-consistent.

Theorem 2: Strengthening

Given a graph $G = (V, E)$ and a labelling function α^i , the function α^{i+1} is stronger than α^i , i.e.,

$$\forall (u, v) \in V^2, \alpha_G^i(u) \neq \alpha_G^i(v) \Rightarrow \alpha_G^{i+1}(u) \neq \alpha_G^{i+1}(v)$$

Theorem 3: Existency of a fixed-point

Given a graph $G = (V, E)$, $\exists k \in \mathbb{N}^*$ such that $\forall (u, v) \in V^2, \forall i > 0, \alpha_G^k(u) = \alpha_G^k(v) \Rightarrow \alpha_G^{k+i}(u) = \alpha_G^{k+i}(v)$

Properties of re-labelling

Theorem 1: Correctness

Given an isomorphic-consistent labelling function α^i , the labelling function α^{i+1} is also isomorphic-consistent.

Theorem 2: Strengthening

Given a graph $G = (V, E)$ and a labelling function α^i , the function α^{i+1} is stronger than α^i , i.e.,

$$\forall (u, v) \in V^2, \alpha_G^i(u) \neq \alpha_G^i(v) \Rightarrow \alpha_G^{i+1}(u) \neq \alpha_G^{i+1}(v)$$

Theorem 3: Existency of a fixed-point

Given a graph $G = (V, E)$, $\exists k \in \mathbb{N}^*$ such that $\forall (u, v) \in V^2, \forall i > 0, \alpha_G^k(u) = \alpha_G^k(v) \Rightarrow \alpha_G^{k+i}(u) = \alpha_G^{k+i}(v)$

Properties of re-labelling

Theorem 1: Correctness

Given an isomorphic-consistent labelling function α^i , the labelling function α^{i+1} is also isomorphic-consistent.

Theorem 2: Strengthening

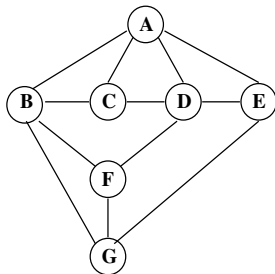
Given a graph $G = (V, E)$ and a labelling function α^i , the function α^{i+1} is stronger than α^i , i.e.,

$$\forall (u, v) \in V^2, \alpha_G^i(u) \neq \alpha_G^i(v) \Rightarrow \alpha_G^{i+1}(u) \neq \alpha_G^{i+1}(v)$$

Theorem 3: Existency of a fixed-point

Given a graph $G = (V, E)$, $\exists k \in \mathbb{N}^*$ such that $\forall (u, v) \in V^2, \forall i > 0, \alpha_G^k(u) = \alpha_G^k(v) \Rightarrow \alpha_G^{k+i}(u) = \alpha_G^{k+i}(v)$

Complete example



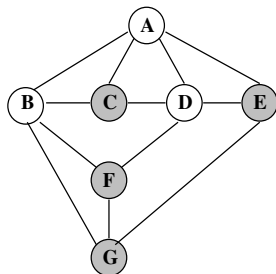
$$\alpha_G^0 : A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1 : \begin{array}{l} A, B, D \rightarrow \emptyset.\{(4, \emptyset)\} \Rightarrow \bigcirc \\ C, E, F, G \rightarrow \emptyset.\{(3, \emptyset)\} \Rightarrow \end{array}$$

$$\alpha_G^2 : \begin{array}{l} A \rightarrow \bigcirc.\{(2, \bigcirc), (2,)\} \\ E, F \rightarrow .\{(2, \bigcirc), (1,)\} \\ B, D \rightarrow \bigcirc.\{(1, \bigcirc), (3,)\} \\ C \rightarrow .\{(3, \bigcirc)\} \\ G \rightarrow .\{(1, \bigcirc), (2,)\} \end{array}$$

$$\alpha_G^3 : \begin{array}{l} E \rightarrow .\{(1,), (1,), (1,)\} \\ F \rightarrow .\{(2,), (1,)\} \\ B \rightarrow .\{(1,), (2,), (1,), (1,)\} \\ D \rightarrow .\{(1,), (1,), (2,)\} \end{array}$$

Complete example



$$\alpha_G^0 : A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1 : \begin{array}{l} A, B, D \rightarrow \emptyset.\{(4, \emptyset)\} \Rightarrow \bigcirc \\ C, E, F, G \rightarrow \emptyset.\{(3, \emptyset)\} \Rightarrow \bullet \end{array}$$

$$A \rightarrow \bigcirc.\{(2, \bigcirc), (2,)\}$$

$$E, F \rightarrow .\{(2, \bigcirc), (1,)\}$$

$$\alpha_G^2 : B, D \rightarrow \bigcirc.\{(1, \bigcirc), (3,)\}$$

$$C \rightarrow .\{(3, \bigcirc)\}$$

$$G \rightarrow .\{(1, \bigcirc), (2,)\}$$

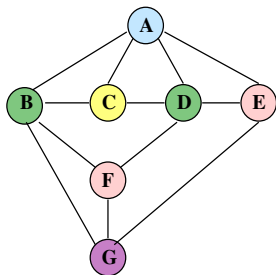
$$E \rightarrow .\{(1,), (1,), (1,)\}$$

$$F \rightarrow .\{(2,), (1,)\}$$

$$\alpha_G^3 : B \rightarrow .\{(1,), (2,), (1,), (1,)\}$$

$$D \rightarrow .\{(1,), (1,), (2,)\}$$

Complete example



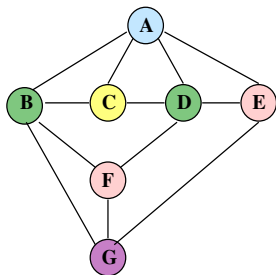
$$\alpha_G^0 : A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1 : \begin{array}{l} A, B, D \rightarrow \emptyset.\{(4, \emptyset)\} \Rightarrow \bigcirc \\ C, E, F, G \rightarrow \emptyset.\{(3, \emptyset)\} \Rightarrow \bigcirc \end{array}$$

$$\alpha_G^2 : \begin{array}{l} A \rightarrow \bigcirc.\{(2, \bigcirc), (2, \bullet)\} \Rightarrow \bigcirc \\ E, F \rightarrow \bullet.\{(2, \bigcirc), (1, \bullet)\} \Rightarrow \bigcirc \\ B, D \rightarrow \bigcirc.\{(1, \bigcirc), (3, \bullet)\} \Rightarrow \bullet \\ C \rightarrow \bullet.\{(3, \bigcirc)\} \Rightarrow \bullet \\ G \rightarrow \bullet.\{(1, \bigcirc), (2, \bullet)\} \Rightarrow \bullet \end{array}$$

$$\alpha_G^3 : \begin{array}{l} E \rightarrow \cdot.\{(1, \cdot), (1, \cdot), (1, \cdot)\} \\ F \rightarrow \cdot.\{(2, \cdot), (1, \cdot)\} \\ B \rightarrow \cdot.\{(1, \cdot), (2, \cdot), (1, \cdot), (1, \cdot)\} \\ D \rightarrow \cdot.\{(1, \cdot), (1, \cdot), (2, \cdot)\} \end{array}$$

Complete example



$$\alpha_G^0 : A, B, C, D, E, F, G \rightarrow \emptyset$$

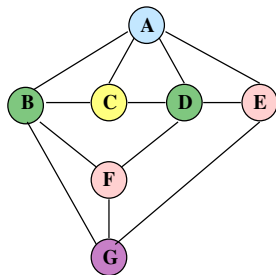
$$\alpha_G^1 : \begin{array}{l} A, B, D \rightarrow \emptyset.\{(4, \emptyset)\} \Rightarrow \bigcirc \\ C, E, F, G \rightarrow \emptyset.\{(3, \emptyset)\} \Rightarrow \bigcirc \end{array}$$

$$\alpha_G^2 : \begin{array}{l} A \rightarrow \bigcirc.\{(2, \bigcirc), (2, \bullet)\} \Rightarrow \text{light blue} \\ E, F \rightarrow \bullet.\{(2, \bigcirc), (1, \bullet)\} \Rightarrow \text{pink} \\ B, D \rightarrow \bigcirc.\{(1, \bigcirc), (3, \bullet)\} \Rightarrow \text{green} \\ C \rightarrow \bullet.\{(3, \bigcirc)\} \Rightarrow \text{yellow} \\ G \rightarrow \bullet.\{(1, \bigcirc), (2, \bullet)\} \Rightarrow \text{purple} \end{array}$$

\rightsquigarrow relabel E, F, B and D

$$\alpha_G^3 : \begin{array}{l} E \rightarrow .\{(1,), (1,), (1,)\} \\ F \rightarrow .\{(2,), (1,)\} \\ B \rightarrow .\{(1,), (2,), (1,), (1,)\} \\ D \rightarrow .\{(1,), (1,), (2,)\} \end{array}$$

Complete example



$$\alpha_G^0: A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1: \begin{array}{l} A, B, D \rightarrow \emptyset.\{(4, \emptyset)\} \Rightarrow \circ \\ C, E, F, G \rightarrow \emptyset.\{(3, \emptyset)\} \Rightarrow \bullet \end{array}$$

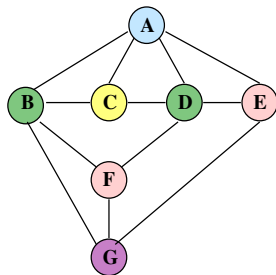
$$\alpha_G^2: \begin{array}{l} A \rightarrow \circ.\{(2, \circ), (2, \bullet)\} \Rightarrow \circ \\ E, F \rightarrow \bullet.\{(2, \circ), (1, \bullet)\} \Rightarrow \circ \\ B, D \rightarrow \circ.\{(1, \circ), (3, \bullet)\} \Rightarrow \bullet \\ C \rightarrow \bullet.\{(3, \circ)\} \Rightarrow \bullet \\ G \rightarrow \bullet.\{(1, \circ), (2, \bullet)\} \Rightarrow \bullet \end{array}$$

\rightsquigarrow relabel E, F, B and D

$$\alpha_G^3:$$

$$\begin{array}{l} E \rightarrow \circ.\{(1, \circ), (1, \bullet), (1, \bullet)\} \\ F \rightarrow \circ.\{(2, \bullet), (1, \bullet)\} \\ B \rightarrow \bullet.\{(1, \circ), (2, \bullet), (1, \circ), (1, \bullet)\} \\ D \rightarrow \bullet.\{(1, \circ), (1, \bullet), (2, \circ)\} \end{array}$$

Complete example



$$\alpha_G^0: A, B, C, D, E, F, G \rightarrow \emptyset$$

$$\alpha_G^1: \begin{array}{l} A, B, D \rightarrow \emptyset.\{(4, \emptyset)\} \Rightarrow \circ \\ C, E, F, G \rightarrow \emptyset.\{(3, \emptyset)\} \Rightarrow \bullet \end{array}$$

$$\alpha_G^2: \begin{array}{l} A \rightarrow \circ.\{(2, \circ), (2, \bullet)\} \Rightarrow \circ \\ E, F \rightarrow \bullet.\{(2, \circ), (1, \bullet)\} \Rightarrow \circ \\ B, D \rightarrow \circ.\{(1, \circ), (3, \bullet)\} \Rightarrow \bullet \\ C \rightarrow \bullet.\{(3, \circ)\} \Rightarrow \bullet \\ G \rightarrow \bullet.\{(1, \circ), (2, \bullet)\} \Rightarrow \bullet \end{array}$$

\rightsquigarrow relabel E, F, B and D

$$\alpha_G^3: \begin{array}{l} E \rightarrow \circ.\{(1, \circ), (1, \bullet), (1, \bullet)\} \\ F \rightarrow \circ.\{(2, \bullet), (1, \bullet)\} \\ B \rightarrow \bullet.\{(1, \circ), (2, \bullet), (1, \circ), (1, \bullet)\} \\ D \rightarrow \bullet.\{(1, \circ), (1, \bullet), (2, \circ)\} \end{array}$$

Every vertex has a different label \Rightarrow stop

Iterated Local Label (ILL) consistency

ILL-consistency of a GIP constraint

The domain of a variable associated with a vertex labelled by l only contains values also labelled by l

ILL-filtering

- Start from α^0 that labels every vertex with \emptyset
- Repeat:
 - Compute α^{i+1} from α^i
 - Rename labels (same renaming for both graphs)
 - Remove non consistent labels from variable domains
- Until a fixed point is reached or a domain becomes empty

Time complexity of ILL-filtering

- Each relabelling step $\rightsquigarrow \mathcal{O}(|E|)$
- $|V|$ relabelling steps in the worst case

Experimental evaluation of ILL-consistency

Question

How often ILL-filtering detects inconsistencies ?

Experiments on graphs with 7 vertices

- Generate the set \mathcal{G}_7 of all different graphs with 7 vertices
 $\rightsquigarrow |\mathcal{G}_7| = 853$
- Consider every pair $(G, G') \in \mathcal{G}_7$ such that $G \neq G'$
 $\rightsquigarrow 363,378$ pairs of non isomorphic graphs
- Count the number of times ILL-filtering detects an inconsistency

detects inconsistency	does not detect
363,361	17

Experimental evaluation of ILL-consistency

Question

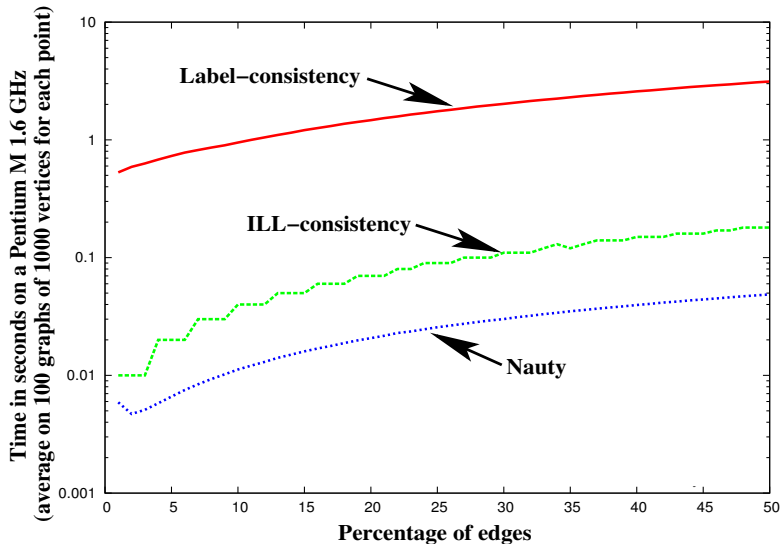
How many different labels, and how many relabelling steps ?

Experiments on non-automorphic graphs

- Generate 100 graphs with k vertices (edge density = 1%)
 \rightsquigarrow non-automorphic graphs only
- Perform ILL-filtering and count the number of different labels
 $\#labels = k \Rightarrow$ filtering solves any GIP involving this graph

k	ILL-consistency	
	#labels	#steps
200	199.64	3.40
400	400.00	2.88
600	600.00	2.14
800	800.00	2.01

Scale-up properties w.r.t. edge density



Scale-up properties w.r.t. the number of vertices

