Problème de satisfaction de Contrainte (CSP)

Master 1 SID

MOP

Plan

- Applications,
- Définition,
- Formalisation (problème des 4 reines),
- Résolution
  - Résolution par exploration,
  - Méthode génère et test,
  - Algorithme avec backtracking,
  - Heuristiques génériques,
  - Filtrage: Forward checking,
  - discussion2

Domaines dôapplication

- Problème dôaffectation, (qui enseigne quoi),
- Emploi du temps (un cours est affecté à quelle salle et à quelle tranche horaire)
- Le problème dôordonnancement de transport,
- Configuration matériel,
- Ordonnancement dôun montage dôusineé

Introduction

- La résolution de problème par exploration dans un espace dœtat nécessitent:
  - Des heuristiques spécifiques au problème
  - Les états sont accessibles par des routines spécifiques au problème (fonction successeur, la fonction heuristique et le test de lœtat final)

CSP

- Dans un CSP:
  - Les états et le test de l'état but se conforment à une représentation standard,
  - En conséquence les algorithmes sont simples et mettent en œuvres des heuristiques génériques non spécifiques au problème

Définition

- Un CSP est défini par:
  - Un ensemble de variables X1, X2,… , Xn et
  - Un ensemble de contraintes C1, C2,… , Cm
  - Chaque variable Xi a un domaine Di
  - Chaque contrainte Cj comprend un sous-ensemble de variables et spécifie le combinaisons possibles des valeurs de sous-ensemble

Contrainte

- Une contrainte est une relation logique (une propriété qui doit être vérifiée) entre différentes inconnues, appelées variables, chacune prenant ses valeurs dans un ensemble donné, appelé domaine.
- Exemple: "*x + 3*y = 12*" , "*x - 2*y = z*"

Arité d'une contrainte

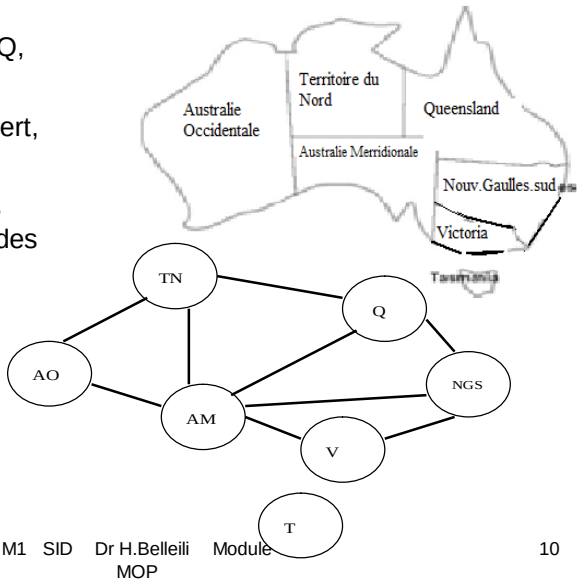
- L'arité d'une contrainte est le nombre de variables sur lesquelles elle porte.
  - Contrainte **unaire**: ne porte que sur une variable. Exemple : *x*x = 4* , *est-un-triangle(y)*
  - ***binaire*** : met en relation 2 variables. Exemple "*x Î y*" ou encore "*A U B = A*"
  - ***ternaire*** : met en relation 3 variables. Exemple "*x+y < 3*z-4*" ou encore "*(non x) ou y ou z = vrai*"
  - **n-aire**: met en relation un ensemble de *n* variables: exemple: "*toutesDifférentes(E)*", où *E* est un ensemble de variables, qui contraint toutes les variables appartenant à *E* à prendre des valeurs différentes
- Lorsqu'un CSP ne contient que des contraintes binaires on dit que le CSP est binaire

Etat du problème

- État du problème: une affectation de valeurs à tout ou partie de variables {Xi=vi,Xj=vjé }
- Une affectation qui ne viole aucune contrainte est dite consistante
- Affectation complète si elle affecte toutes les variables
- Un solution est une affectation complète qui satisfait toutes les contraintes
- Certains CSP exigent en plus la maximisation dûne fonction objective

Exemple: coloration dûne carte

- **Variables**: AO, TN, AM, Q, NGS, V, T
- **Domaines** : D={rouge, vert, bleu}
- **Contraintes**: les régions adjacentes doivent avoir des couleurs différentes.
  - AO<>TN
  - Ou, (AO,TN) ∈ {(rouge,vert), (Rouge,bleu), etc}



Coloration dûune carte

Solution: Une affectation qui satisfait toutes les contraintes: {*AO = rouge*, *TN = vert*, *AM = bleu Q = rouge NGS = vert V rouge*, *T = vert*} en est une

Types de CSP

Variables discrètes

- ï Domaines finis; nombre de variables *n* et taille max du domaine de nûimporte quelle variable est *d*; alors O(d^n) affectations complètes.
- ï Domaines infinis (entiers, chaînes de caractères, etc.)
  - Å Ex: Ordonnancement de tâches; les variables sont les jours de début et de fin des tâches.
  - Å Demande un langage de contraintes, ex. *débutTâche1* + 5 < *débutTâche3*
  - ÅContraintes linéaires: soluble; contraintes non linéaires: non décidable

Å **Variables continues**

- ï Ex: temps de début et de fin des observations dûun télescope
- ï Contraintes linéaires: La catégorie de CSP à domaines continus la mieux connue est celle de la programmation linéaire où les contraintes sont des inégalités linéaires soluble en temps polynomial par rapport au nombre de variables.

Å CSP booléens: dont les variables valent soit vrai soit faux.

Modélisation dôun CSP

- La modélisation dôun problème en CSP revient à
  - identifier toutes les variables X du problème (inconnues du problème)
  - la fonction D qui associe à chaque variable son domaine (les valeurs que la variable peut prendre)
  - Identifier les contraintes C entre les variables
- On peut modéliser un problème donné par différents CSPs
- La modélisation choisie pet avoir une influence sur lôefficacité de la résolution

Exemple : le problème des reines

- Il existe plusieurs modélisations du problème en un CSP.
- Nous en donnons trois modélisations possibles

Modélisation 1: le problème des reines

- Les inconnues du problème sont les positions des reines sur l'échiquier
- La position d'une reine i est déterminée par un numéro de ligne li et un numéro de colonne ci
- Les contraintes : les reines doivent être sur des lignes différentes et des colonnes différentes et des diagonales différentes
- Questions:
  1. Donner les contraintes qui permettent de vérifier si deux reines ne sont pas sur la même diagonale?
  2. Formaliser le problème des 4 reines sur un échiquier de 4x4 selon cette modélisation.

Modélisation 2:

- Puisque l'on sait qu'il y a une reine par colonne, le problème peut se résumer à déterminer sur quelle ligne se trouve la reine placée sur la colonne i
- 2ième modélisation: associer une variable i à chaque colonne i de telle sorte que Xi désigne le numéro de la ligne sur laquelle placer la reine de la colonne i
- Question: Donner la deuxième modélisation pour n=4

Modélisation 3

- Une autre façon, complètement opposée, de modéliser le problème consiste à choisir comme variables non pas les position des reines, mais les états des cases de l'échiquier
- Soit Xij la variable associée à chaque case située à la ligne i et la colonne j
- Chaque variable peut prendre pour valeur 0 (case vide) ou 1 (s'il y a une reine sur la case)
- Question: donner une formalisation du problème selon cette modélisation

Les reines discussion

- Question: quelle est la meilleure modélisation?
- Il y a au moins trois réponses à cette question: la meilleure modélisation est
  - Celle qui modélise le mieux la réalité, de ce point de vue les 3 modélisation sont équivalentes
  - Celle qui est la plus facile à trouver, la première modélisation est la plus simple
  - Celle qui permettra de résoudre le problème plus efficacement. On verra que la 2ième modélisation est la meilleure dès lors qu'elle met en jeux moins de variables que les 2 autres modélisations. Ce qui signifie qu'elle sera de complexité nettement inférieure aux autres.

Résolution d'un CSP

- Formulation d'un CSP sous forme d'un problème de recherche standart
- L'algorithme "génère et test "
- L'algorithme "simple retour-arrière"
- L'algorithme "anticipation " (forward checking)
- Intégration d'heuristiques

Formulation de recherche standard

- États: les valeurs assignées jusqu'à maintenant
- État initial: l'affectation vide {}
- Fonction successeurs: Affecter une valeur à une variable non affectée qui ne crée pas de conflit
- Test de but: l'affectation est complète
- Coût du chemin: un coût constant à toutes les étapes

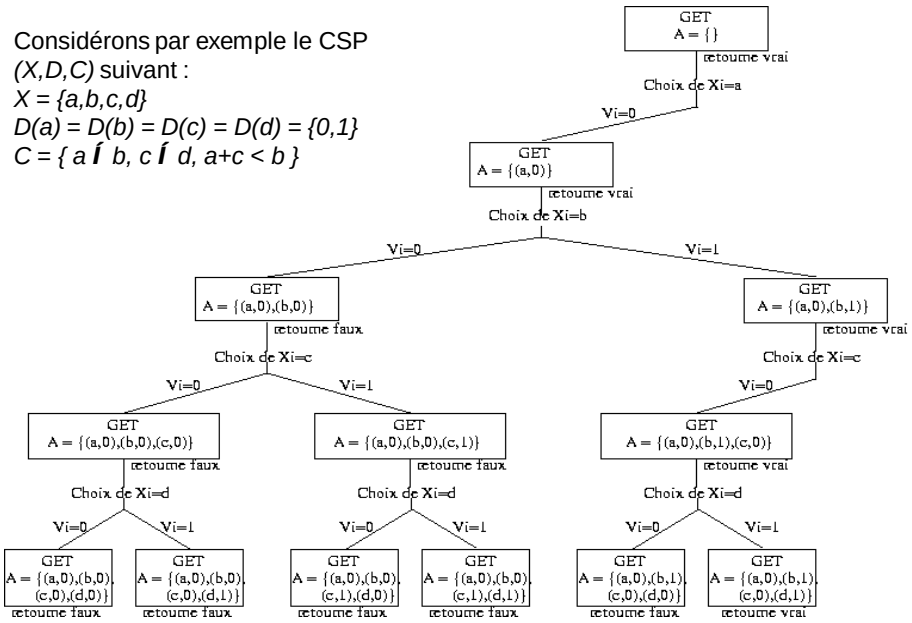
Formulation de recherche standard

- Chaque solution doit être une affectation complète et apparaît à la profondeur *n* si il y a *n* variables
- Le chemin qui a permis d'obtenir la solution n'a pas d'importance

Génère et Test

- Énumère toutes les affectations totales possibles jusqu'à en trouver une qui satisfasse toutes les contraintes

Considérons par exemple le CSP *(X,D,C)* suivant :
X = {a,b,c,d}
D(a) = D(b) = D(c) = D(d) = {0,1}
C = { a Í b, c Í d, a+c < b }



Critique

- l'espace de recherche est souvent de taille tellement importante que l'algorithme "génère et test" ne pourra se terminer en un temps raisonnable:
- Soit un CSP *(X,D,C)* comportant *n* variables (*X = {X1, X2, ..., Xn}*
- Le domaine est défini par *E = { {(X1,v1), (X2,v2), ..., (Xn,vn)} / quelque soit i compris entre 1 et n, vi est un élément de D(Xi)*
- et le nombre d'éléments de cet espace de recherche est défini par *| E | = | D(X1) | * | D(X2) | * ... * | D(Xn) |* de sorte que, si tous les domaines des variables sont de la même taille *k* (autrement dit, *|D(Xi)| =k*), alors la taille de l'espace de recherche est *| E | = k^n*
- *le nombre d'affectations à générer croit de façon exponentielle en fonction du nombre de variables du problème*

Améliorations

- ne développer que les affectations partielles consistantes : dès lors qu'une affectation partielle est inconsistante, il est inutile de chercher à l'étendre en une affectation totale puisque celle-ci sera nécessairement inconsistante ;
  Cette idée est reprise dans l'algorithme "simple retour-arrière"
- réduire les tailles des domaines des variables en leur enlevant les valeurs "incompatibles" : pendant la génération d'affectations, on filtre le domaine des variables pour ne garder que les valeurs "localement consistantes" avec l'affectation en cours de construction, et dès lors que le domaine d'une variable devient vide, on arrête l'énumération pour cette affectation partielle ;
  Cette idée est reprise dans l'algorithme "anticipation" (forward checking)

Exploration avec retour arrière

- revient à faire une exploration en profondeur qui choisit une valeur pour une variable à la fois et remonte dans l'arbre (backtracks) lorsqu'il reste une variable à laquelle on ne peut pas affecter de valeur légale
- Complexité: algorithme non informé → non efficace pour les grandes tailles de problèmes

fonction Exploration-backtracking (*csp*) **retourner** une solution ou echec

retourner backtracking-récursive (*{ }*, *csp*)

fonction backtracking-récursive (*affectation*, *csp*) **retourner** une solution ou echec

Si *affectation* complète **alors retourner** *affectation*

var := select-variable-non-affecté (variable[*csp*], *affectation, csp*)

Pour chaque *valeur* **dans** ordonner-valeurs-domaine (*var*, *affectation,csp*) faire

Si *valeur* est consistante avec *affectation* conformément à contraintes (csp) alors

Ajouter {*var=valeur*} à *affectation*

résultat:=backtracking-récursive(*affectation*, *csp*)

Si *résultat*<> *echec* **alors *retourner*** *résultat*

Supprimer *var=valeur* de *affectation*

Retourner *echec*

Les fonctions select-variable-non affecté et ordonner-valeurs-domaine peuvent servir à implémenter les heuristiques génériques

Améliorer lôefficacité de la recherche

- Quelle variable devrait être assignée et dans quel ordre ses valeurs devraient être essayées ?
- Quelles sont les implications des affectations de la variable courante sur les autres variables non affectées ?
- Lorsquôun chemin échoue, est-ce que la recherche pourrait éviter cet échec dans les chemins suivants ?

Heuristiques du CSP

- les algorithmes de résolution du CSP mettent en î uvres des heuristiques génériques non spécifiques au problème:
  - Heuristiques sur lôordre dans lequel les variables sont choisies
  - Heuristique sur lôordre dans lequel les valeurs seront choisi

Ordre des variables

On dispose de 2 heuristiques:

- Heuristique du nombre de valeurs restantes minimum (minimum remaining values (MRV)). Appelée aussi *fail first.*
  - Principe: Choisir la variable avec le moins de valeurs possibles.
- Heuristique du degré
  - Choisir la variable présente dans le plus de contraintes. Utilisée pour départager des choix lorsque nous sommes confrontés à des variables avec le même MRV.

  Exemple: dans lôexemple de coloriage la variable AM est la variable de degré le plus élevée

Ordre des valeurs

- Lorsqu'une variable a été sélectionnée, il faut déterminer l'ordre dans lequel examiner ses valeurs
  - On utilise l'Heuristique des valeurs les moins contraignantes (leastconstraining-value): Choisir la valeur qui va enlever le moins de choix pour les variables voisines.

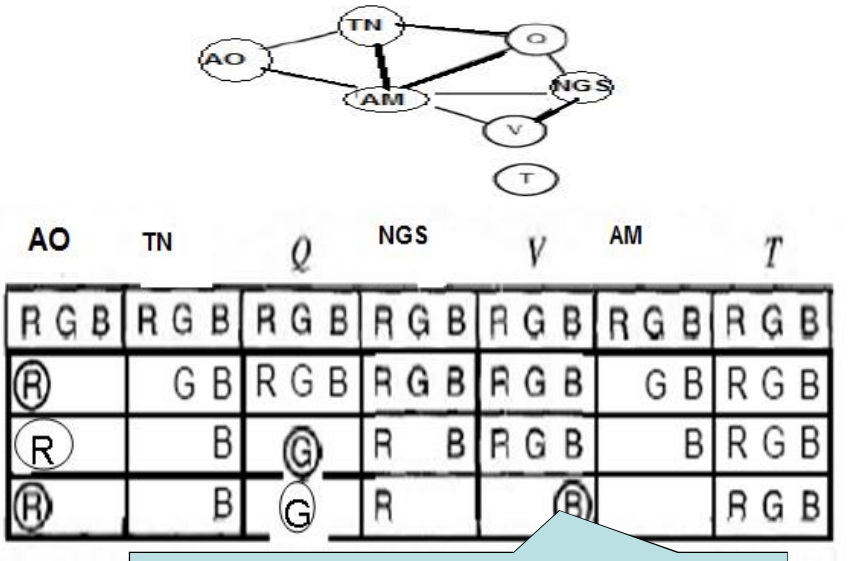
  Exemple: AO=rouge, et TN= vert, si le choix suivant est Q, bleue est un mauvais choix puisqu'il éliminerait la dernière valeur légale restante pour AM le voisin de Q.

Le filtrage

- Pour améliorer la vitesse de l'algorithme backtrack, il existe la méthode par filtrage.
- Le filtrage est d'aller de l'avant de la recherche par retours en arrière et trouver des valeurs dans les domaines des variables non affectées, qui peuvent être éliminés en toute sécurité,
- Il existe deux méthodes de filtrage:
  - Le forward checking,
  - La propagation de contraintes

Algorithme par anticipation (Forward checking)

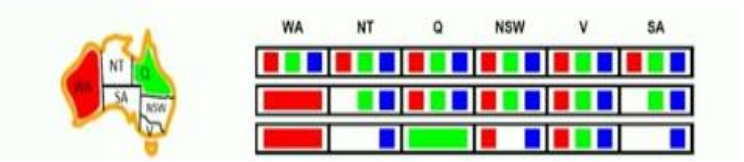
- Appelé vérification en aval:
- Chaque fois quûune variable X est affectée, le processus de forward checking examine chaque variable non affectée Y qui est reliée à X par une contrainte et supprime du domaine de Y toutes les valeurs non consistantes avec la valeur choisie pour X



Le domaine de AM est vide donc le forward checking a detecté que lûaffectation partielle {AO= rouge, Q=vert, V=bleu} est inconsistante donc backtrack

Propagation des contraintes

- Le forward checking est un moyen efficace pour calculer lôheuristique MRV
- Le forward checking détecte de nombreuses inconsistances mais ne les détecte pas toutes



- NT et SA ne peuvent pas être Bleus en même temps,
- Le forward checking ne détecte pas lôinconsistance car il ne pousse pas suffisamment loin vers lôaval
- La propagation de contraintes raisonne de contrainte à contrainte

Consistance dôarc

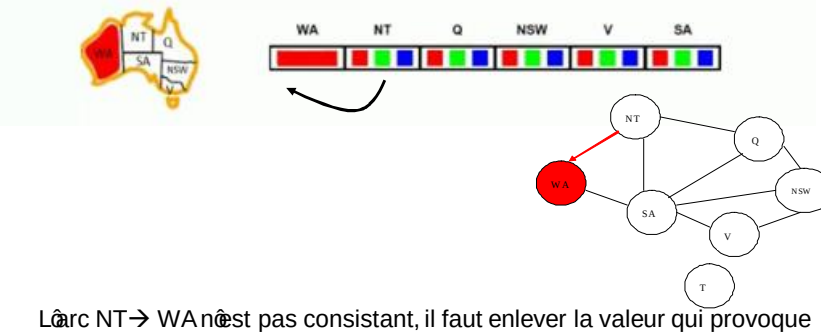
- La propagation de contrainte peut être obtenue par la méthode appelée: renforcement de la consistance dôarc

Consistance dûun arc

- Le graphe de contraintes nêest pas orienté
- Il nôy a pas la notion dâarc
- Donc quêest-ce quêest la consistance dûun arc?
- La notion dâarc est une nouvelle idée qui renforce le filtrage

Consistance dûun arc

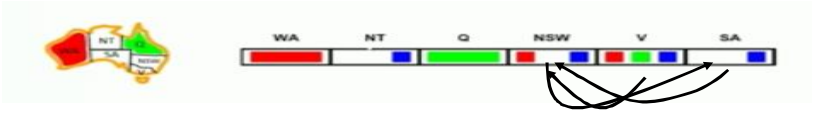
Un arc X→Y est **consistent** ssi pour chaque valeur x (de la queue de lâarc) il existe une valeur y (de la tête de lâarc) qui peut être affectée sans violation de contrainte



Lâarc NT→ WA nêest pas consistant, il faut enlever la valeur qui provoque lôinconsistance de la queue de lâarc

Le forward checking renforce la consistance seulement des arcs pointant (à la variable qui vient dêtre affectée) à chaque nouvelle affectation

Consistance dâarc du graph



Important: si X perd une valeur alors les voisins de X doivent être revérifiés

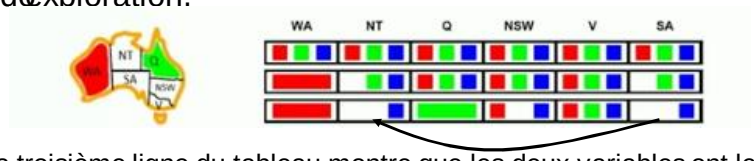
La consistance dâarc détecte lâechec plus tôt que le forward checking

La consistance dâarc peut être exécutée comme un prétraitement ou après chaque affectation

Quel est le côté négatif de la consistance dâarc?

Exemple (suite)

Å On peut aussi appliquer la consistance des arcs à lâarc allant de SA vers NT au même stade du processus dâexploration.



La troisième ligne du tableau montre que les deux variables ont le domaine {*bleu*}.
Il en résulte que la valeur *bleu* doit être supprimé du domaine de SA qui devient vide
Lâapplication de la consistance des arcs a permis de détecter précocement lâinconsistance que le *forward checking* aurait négligée

Définition arc-consistance

- L'arc consistance qui est défini ainsi ; Soit *sx* le domaine courant de la variable X, soit ***c*** une contrainte sur X et Y : la valeur $vx \in sx$ est arc-consistante pour ***c*** si : $\exists$ vy $\in sy$ tel que ($vx; vy$) $\in sol(\boldsymbol{c})$;
- De même on dit que la contrainte *c* est arc-consistante si: $\forall$vx $\in$sx, vx est arc-consistante
- et vice versa en échangeant le rôle de X et de Y.

Consistance des arcs (arc-consistance)

- La vérification de la consistance des arcs peut se faire soit lors d'une étape de prétraitement avant le démarrage du processus d'exploration soit dans le cadre d'une étape de propagation après chaque affectation effectuée pendant l'exploration
- Principe: chaque fois qu'une valeur est supprimée du domaine d'une variable, une nouvelle inconsistance peut survenir dans les arcs qui pointent vers cette variable

Lâalgorithme AC3

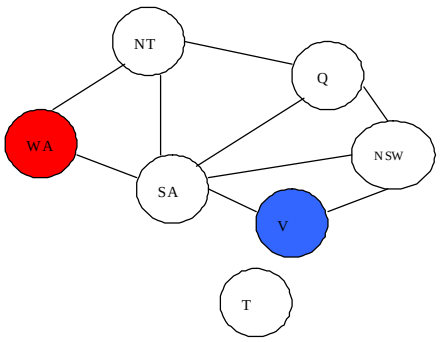
- La version complète de lâarc-consistance est AC-3,
- Lâalgorithme Ac3 utilise une file dâattente.
- Dès quûn domaine est modifié, on ne place dans la file que les arcs pouvant être affectés par la modification.
- Le but étant dâaméliorer la vitesse de convergence vers la stabilité en éliminant les vérifications inutiles.

Mise en î uvre arc-consistance

- On utilise une file dâattente pour stocker les arcs dont il faut vérifier la consistance
- Chaque arc (Xi,Xj) est tour à tour supprimé de cette file et vérifié
- Sûil faut supprimer lûune des valeurs du domaine de Xi, tous les arcs (Xk,Xi) qui pointent vers Xi doivent être réinsérés dans la files afin dâêtre vérifiés

Exemple

- Utiliser l'algorithme AC-3 pour montrer que l'arc-consistance peut détecter l'inconsistance de l'affectation {WA=rouge, V=bleu}



Solution

- Soit l'affectation {WA=rouge, V=bleu} pour un ordre possible des arcs, l'exécution de AC3 se fait ainsi:

SA-WA enlever R de SA {G,B}
SA-V enlever B de SA {G}
NT-WA enlever R de NT {GB}
NT-SA enlever G de TN {B}
NSW-SA enlever G de NSW {R,B}
NSW-V enlever B de NSW {R}
Q-NT enlever B de Q {R,G}
Q-SA enlever G de Q {R}
Q-NSW enlever R de Q { } aucune valeur restante possible pour Q (inconsistance)

AC-3

- L'algorithme AC-3 ne montre pas l'ordre dans lequel les arcs sont examinés
- Donc il s'exécutera de différentes manières selon l'ordre dans lequel se trouve les arcs dans la file

Complexité de AC-3

- On peut montrer que si *d* est la taille maxi des domaines des variables et *n* le nombre de contraintes, alors :
- Une contrainte Ci,j est examinée au plus d fois (car une contrainte est examinée à cause de la suppression d'une valeur dans le domaine de sa seconde variable).
- Il y a donc au plus (d × n) examen de contraintes
- chaque examen de contrainte nécessite d^2 vérifications.
- L'algorithme a donc une complexité en $O(n \cdot d^3)$

Discussions

- On ne peut pas trouver un algorithme en temps polynomial susceptible de décider si un CSP donné est consistant
- En effet, Il existe des cas où l'AC3 ne détecte pas l'inconsistance.
- Exemple: L'affectation partielle {WA=rouge et NSW=rouge} est inconsistante mais AC3 ne la détectera pas. (à vérifier)
- On peut définir des formes de propagation plus fortes avec la notion de k-consistance

Contrainte n-aire et consistance d'arc

- Il est possible d'étendre la notion de cohérence d'arc pour traiter des contraintes n-aires et non simplement binaires
- C'est ce qu'on appelle *cohérence d'arc généralisée* ou encore *hyercohérence d'arc.*
- Une variable Xi est *arc-cohérente généralisé* relativement à une contrainte *n-aire* si, pour chaque valeur *v* du domaine de Xi, il existe des valeurs issues des variables impliquées dans la contrainte telles que celle-ci soit vérifiée.

- Exemple: soit la contrainte X<Y<Z si toutes les variables ont un domaine {0,1,2,3}, alors pour rendre la variable X cohérent avec la contrainte, nous devons éliminer 2 et 3 du domaine de X car la contrainte ne peur être satisfaite lorsque X vaut 2 ou 3.

K-consistance

- Un CSP est K-consistant si pour tout ensemble de k-1 variables et pour toute affectation consistante de ces variables, on peut toujours affecter une valeur consistante à nôimporte quelle k-ième variable
- 1-consistance: chaque variable individuelle est elle-même consistante (consistance des nî uds)
- 2-consistance: identique à lôarc-consistance
- 3-consistance: toute paire de variables adjacentes peut toujours être étendue à une troisième variable voisine (consistance du chemin)

Graphe fortement k-consistant

- Un graphe est fortement k-consistant sôil est k-consistant, (k-1)-consistant, (k-2)-consistant é jusquôà 1-consistant
- Si un CSP comptant n nî uds et qui a été rendu k-consistant (avec k=n) alors un tel CSP peut être résolu sans backtracking:
- Pour commencer on choisit une valeur consistante pour X1, on est assuré de trouver une valeur pour X2 (2-consistance), ensuite une valeur pour X3 (3-consistance)é
- Complexité $O(n^d)$: tout algorithme qui établit la n-consistance doit prendre un temps exponentielle en n.
- Solution intermédiaire: trouver la plus petite valeur de k tel que lôexécution de la k-consistance assure de résoudre le CSP sans Backtracking (méthode impraticable)
- En pratique la détermination du niveau approprié de la vérification de la consistance est essentiellement empirique (expérimentale)

Backtracking intelligent

Backtracking simple, lorsqu'une branche d'exploration échoue, le point de décision le plus récent est révisé

- Exemple: Soit l'affectation {Q=rouge, NSW=vert, V=bleu, T=rouge}, lorsque l'on essaye la variable suivante SA on s'aperçoit que toutes les valeurs violent une contrainte. On remonte alors à T et on essaye une nouvelle couleur→ une pure perte de temps car changer la couleur de T ne résoud pas le problème

Backracking intelligent: Il faut remonter vers l'ensemble des variables qui est à l'origine de l'échec appelé *ensemble de conflit*

- Exemple: Dans l'exemple l'ensemble de conflit de SA est {Q,NSW,V}

Ensemble de conflits

- L'ensemble de conflits d'une variable X est l'ensemble des variables précédemment affectées qui sont reliées à X par des contraintes
- La méthode Backjumping remonte à la variable la plus récente dans l'ensemble de conflit (dans l'exemple c'est la variable V)

Ensemble de conflits (suite)

- On modifie l'exploration backtracking de manière à stocker l'ensemble de conflits lorsqu'il recherche une valeur légale à affecter. S'il ne retrouve pas de valeurs légales, il doit retourner à l'élément le plus récent de l'ensemble de conflits.

Backjumping vs forward checking

- Le backjumping se produit lorsque toutes les valeurs d'un domaine sont en conflits avec l'affectation courante
- Le forward checking détecte cet évènement et empêche l'exploration d'atteindre un tel nî ud
- On peut montrer que toutes les branches élaguées par le backjumping sont aussi élaguées par le forward checking
- Donc le backjumping est redondant dans une exploration avec le forward checking

Exploration locale pour CSP

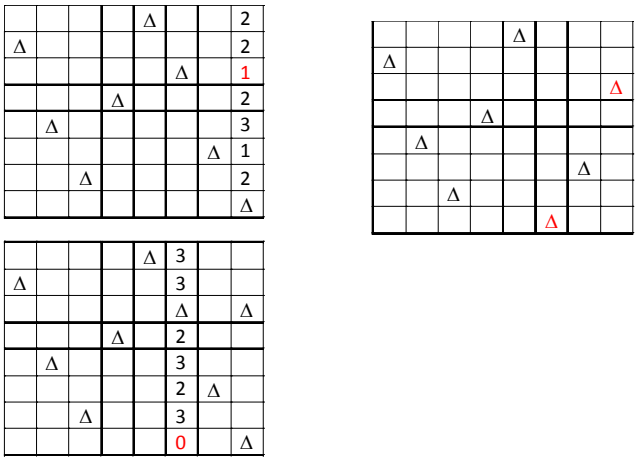
- Très efficace
- Utilise une formulation par états complets:
- L'état initial affecte une valeur pour chaque variable et la fonction successeur procède en modifiant une variable à la fois
- Exemple: les 8 reines. L'état initial peut être une configuration aléatoire et la fonction successeur sélectionne une reine et envisage de la déplacer ailleurs dans sa colonne
- Pour choisir une nouvelle valeur pour une variable, l'heuristique la plus évidente consiste à sélectionner la valeur associée au nombre minimal de conflits avec d'autres variables
- Cette heuristique est appelée MIN-CONFLITS

Algorithme MIN-CONFLIT

Fonction MIN-CONFLITS(*csp*,*max_étapes*) **retourne** une solution ou *echec*
Entrée : *csp*, un problème
max-étapes, le nombre d'étapes autorisées avant de renoncer

Courante:= une affectation complète locale pour *csp*
Pour i= 1 à *max_étapes* **faire**
Si *courante* est une solution pour *csp* alors **retourner** *courante*
var := une variable conflictuelle choisie au hasard dans VARIABLE[*csp*]
valeur := la valeur *v* de *var* qui minimise CONFLITS(*var*,*v*,*courante*,*csp*)
Set *var* = *valeur* dans *courante*

Retourner echec



discussions

- Lôexploration locale est utilisée pour modifier les paramètres à la volée lorsque le problème change.
- Exemple: planification dôune semaine de vols aériens mettant en jeu des milliers de vols et des dizaines de milliers de passagers. Il suffit que le mauvais temps sôabatte sur un aéroport pour que le plan devient inapplicable
- La recherche locale permet de réaménager le plan avec un minimum de changements

Fin