Problèmes de satisfaction de contraintes

IFT-17587

Concepts avancés pour systèmes intelligents

Luc Lamontagne

Plan

- Description des CSP
- Exploration avec retour arrière
- « Forward checking »
- Cohérence des arcs
- Gestion de contraintes spécifiques
- Recherche locale
- Utiliser la structure des problèmes

Problèmes de satisfaction de contraintes

- Définit par:
  - Un ensemble de variables: $X_1, X_2, \ldots, X_n$
  - Un ensemble de contraintes: $C_1, C_2, \ldots, C_m$
- Chaque variable X_i a un domaine non vide de valeurs possibles.
- Un état est une affectation de valeurs pour quelques unes ou toutes les variables.
- Affectation consistante: aucune violation de contraintes.
- Affectation complète: toutes les variables ont une valeur.
- Solution: Affectation complète et consistante.

Exemple: coloration de carte

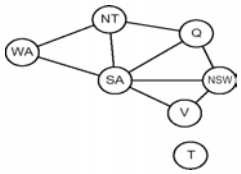
- Variables:
  - *WA*, *NT*, *SA*, *Q*, *NSW*, *V*, *T*
- Domaines:
  - D_i = {*rouge*, *vert*, *bleu*}
- Contraintes: Les régions adjacentes doivent avoir des couleurs différentes.
  - WA différent NT
  - ou, (WA, NT) e {(rouge, vert), (rouge, bleu), etc.}

Exemple: Coloration de carte

- Solution: Une affectation qui satisfait toutes les contraintes:
  - { *WA = rouge*,
    NT = vert,
    SA = bleu,
    Q = rouge,
    NSW = vert,
    V = rouge,
    T = vert }

CSP binaire

- CSP où toutes les contraintes sont reliées à deux variables.
- Peuvent être représentés sous forme de graphe.
- Certains algorithmes généraux de résolution de CSP utilisent la structure de graphe pour accélérer la recherche.
  - Ex: Tasmania est un sous problème indépendant.



Types de CSP

- Variables discrètes
  - Domaines finis :
    - Nombre de variables *n* et *d* valeurs => O(d^n) affectations complètes.
    - CSP booléens.
  - Domaines infinis (entiers, chaînes de caractères, etc.)
    - Ex: Ordonnancement de tâches
      - Les variables sont les jours de début et de fin des tâches.
    - Demande un langage de contraintes
      - Ex. $débutTâche_1 + 5 < débutTâche_3$
    - Contraintes linéaires
      - Soluble.
    - Contraintes non linéaires
      - non décidable.

Types de CSP

- Variables continues
  - Ex: temps de début et de fin des observations du télescope Hubble
  - Contraintes linéaires
    - Soluble en temps polynomial par rapport au nombre de variables.

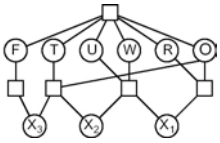
Types de contraintes

- Unaire: Les contraintes ne concernent qu'une variable.
  - Ex: *SA* différent de *vert*
- Binaire: Les contraintes concernent deux variables.
  - Ex: *SA* différent de *WA*
- Ordre plus élevé: Les contraintes concernent 3 variables ou plus.
  - Ex: cryptarithme
- Contraintes de préférence
  - Ex: *rouge* est mieux que *vert*
  - Souvent représenté par un coût pour chaque affectation de variable
  - Problèmes d'optimisation de contraintes

Cryptarithme

- Variables: F, T, U, W, R, O, X_1, X_2, X_3
- Domaines: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Contraintes:
  - Différentes(F, T, U, W, R, O)
  - $O + O = R + 10 * X_1$
  - $X_1 + W + W = U + 10 * X_2$
  - $X_2 + T + T = O + 10 * X_3$
  - $X_3 = F$

```
  T W O
+ T W O
-------
F O U R
```



Applications des CSP

- Problèmes d'assignation
  - Ex: Qui enseigne quel cours?
- Problèmes d'horaire
  - Ex: Quelle classe est offerte, quand et où?
- Planification pour le transport
- Planification à l'intérieur d'une usine

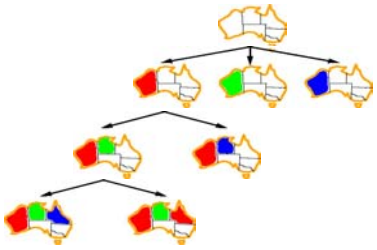
Formulation de recherche standard

- États: les valeurs assignées jusqu'à maintenant
  - État initial: l'affectation vide {}
- Fonction successeurs: Affecter une valeur à une variable non affectées qui ne crée pas de conflit
- Test de but: l'affectation est complète
- Commentaires:
  - Même formulation pour tous les CSP
  - Toutes les solutions à n variables apparaissent à la profondeur n.
    - Profondeur d'abord
  - Le chemin n'est pas important, donc on peut aussi utiliser des algorithmes itératifs.

Exploration avec retour arrière

- L'assignation des variables est commutative
  - Ex: [*WA* = *rouge* suivi de *NT* = *vert*] est la même chose que [*NT* = *vert* suivi de *WA* = *rouge*]
- À chaque nœud, on a seulement besoin de considérer l'affectation d'une seule variable.
  - $b = d$ et il y a d^n feuilles
- La recherche en profondeur d'abord pour les CSP avec l'assignation d'une seule variable à chaque tour est appelé exploration avec retour arrière.
- Le retour arrière s'effectue lorsqu'il n'y a plus d'affectations possibles.

Exemple retour arrière



Exploration avec retour arrière

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING({}, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(Variables[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to Constraints[csp] then
            add { var = value } to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```


Exploration avec retour arrière :

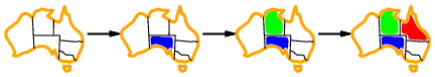
Améliorer l'efficacité

- Quelle variable devrait être assignée au prochain niveau ?
- Dans quel ordre les valeurs devraient être essayées ?
- Quelles sont les implications des affectations de la variable courante sur les autres variables non affectées ?
- Lorsqu'un chemin échoue, est-ce que l'exploration pourrait éviter de reproduire cet échec dans les chemins suivants ?

Ordre des variables

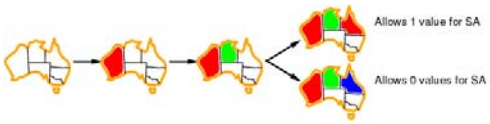
- Choix des variables:
  - Heuristique du nombre minimum de valeurs restantes (M*inimum remaining values* - MRV).
    - Choisir la variable ayant le moins de valeurs possibles.

  - Heuristique du degré
    - Choisir la variable présente dans le plus de contraintes.



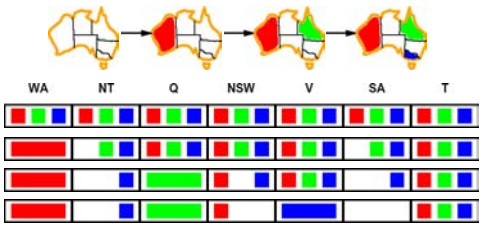
Ordre des valeurs

- La valeur la moins contraignante
  - *Least constraining value*
    - Choisir la valeur qui élimine le moins de choix pour les variables voisines.



Forward checking

- Maintient en tout temps, toutes les valeurs possibles pour chacune des variables.



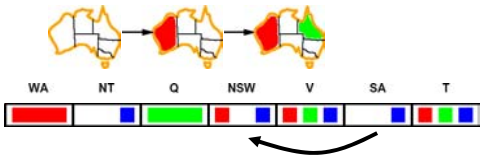
Propagation de contraintes

- Le « *forward checking* » ne permet pas de détecter tous les problèmes.

- NT et SA ne peuvent pas tous les deux être bleus!
- La propagation de contraintes renforce rapidement les contraintes locales.

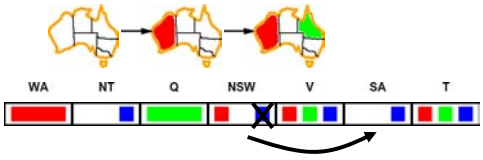
Cohérence des arcs

- Un arc entre *X* et *Y* est cohérent si pour toutes les valeurs *x* de *X*, il y a au moins une valeur possible *y* de *Y*.



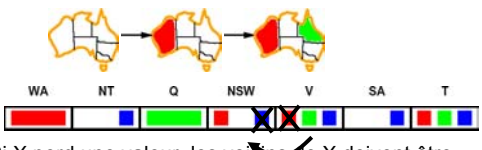
Cohérence des arcs

- Un arc entre *X* et *Y* est cohérent si pour toutes les valeurs *x* de *X*, il y a au moins une valeur possible *y* de *Y*.



Cohérence des arcs

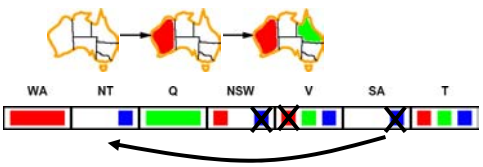
- Un arc entre *X* et *Y* est cohérent si pour toutes les valeurs *x* de *X*, il y a au moins une valeur possible *y* de *Y*.



- Si X perd une valeur, les voisins de X doivent être revérifiés

Cohérence des arcs

- Un arc entre *X* et *Y* est cohérent si pour toutes les valeurs *x* de *X*, il y a au moins une valeur possible *y* de *Y*.



Cohérence des arcs

- Détecte les erreurs plus tôt que le « *forward checking* »
- Peut être exécuté comme :
  - un pré-processus, ou
  - après chaque assignation.

Gestion de contraintes spécifiques

- Certains types de contraintes surviennent souvent dans les problèmes réels
  - On peut donc utiliser des algorithmes spécifiques à ces types de contraintes.
  - Plus performant que les algorithmes généraux.

Gestion de contraintes spécifiques

- Contrainte *toutes différentes*
  - Toutes les variables doivent avoir des valeurs distinctes.
  - S'il y a m variables, n valeurs possibles et $m > n$, alors les contraintes ne peuvent pas être satisfaites.
  - Algorithme:
    - Enlever toutes les variables qui n'ont qu'une valeur dans leur domaine et enlever cette valeur de toutes les autres variables restantes.
    - Continuer tant qu'il y a des variables avec un domaine contenant qu'une seule valeur.
    - Si à un domaine vide est produit, ou qu'il y a plus de variables que de valeurs disponibles, alors une incohérence a été détectée.

Gestion de contraintes spécifiques

- Contrainte sur les ressources
  - Limite sur le nombre de ressources.
  - Par exemple:
    - Il y a un maximum de 10 personnes pour effectuer 4 tâches.
    - Si le domaine de chacune des tâches est {3, 4, 5, 6}, alors les contraintes ne pourront pas être satisfaite.
    - On peut aussi réduire les domaines de valeurs. Par exemple, si le domaine des tâches est {2, 3, 4, 5, 6}, alors les valeurs 5 et 6 peuvent être enlevées des domaines.

Gestion de contraintes spécifiques

- Propagation de bornes
  - Lorsque les domaines sont plus grands, on utilise la propagation de bornes. Par exemple:

Capacité: 165 passagers
Domaine: [0, 165]

Contrainte: 420 passagers à transporter

Capacité: 385 passagers
Domaine: [0, 385]

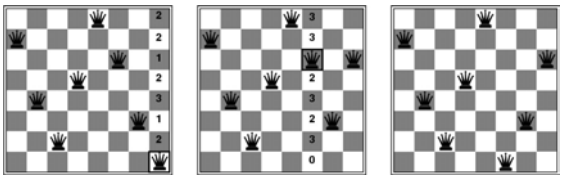
Nouveau domaine: [35, 165]

Nouveau domaine: [255, 385]

Exploration locale pour CSP

- Fonctionnement
  - On commence en assignant une valeur à chacune des variables.
  - Ensuite, la fonction de successeurs change la valeur d'une variable à la fois.
- Heuristique « *min-conflicts* »
  - Choisir la valeur qui cause le moins de conflits
- Utile pour les problèmes « *online* ».

Exploration locale pour CSP



- Avec *min-conflicts*
  - On peut résoudre en moyenne 1 millions de reines en 50 étapes.
- Télescope Hubble
  - Le temps de planification d'une semaine d'activité passe de trois semaines à 10 minutes en utilisant min-conflicts!

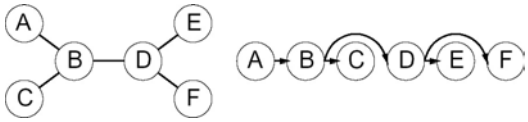
La structure des problèmes CSP

- La structure du graphe des contraintes
  - Utiliser pour trouver des solutions rapidement.
- Composantes connexes
  - Constitué de sous-problèmes indépendants
  - On résout les sous-problèmes indépendamment
  - Supposons que chaque sous-problème contient c variables sur un total de n, alors le coût de la solution en pire cas est de $n/c * d^c$. (linéaire en n)
    - Si n = 80, d = 2 et c = 20
    - d^{80} = 4 milliards d'années à 10 millions de nœuds/sec
    - $4 * 2^{20}$ = 0,4 secondes à 10 millions de nœuds/sec

La structure des problèmes CSP :

Si le graphe est un arbre…

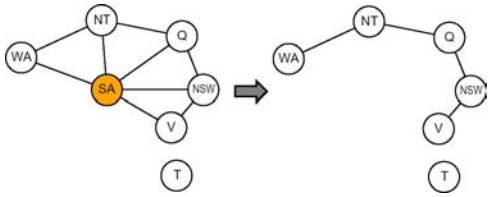
- Si le graphe ne contient pas de cycle, alors le CSP peut être résolut en O(nd^2)
  - Choisir un nœud comme racine et ordonner les autres nœuds de manière à ce que le parent précède toujours.
  - Pour chaque nœud de n à 2, enlever les valeurs incohérentes avec l'arc (X_i, X_j) où X_i est le parent de X_j
  - Pour chaque nœud de 1 à n, affecter une valeur à X_j consistante avec X_i



La structure des problèmes CSP :

Si le graphe n'est pas un arbre…

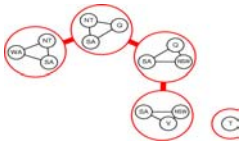
- Mais si la structure est proche d'un arbre
  - on peut fixer la valeur de certains nœuds pour obtenir une structure d'arbre.



La structure des problèmes CSP :

Décomposition en arbre

- Une autre approche pour réduite la complexité du problème
  - Approche diviser pour régner
  - Chaque variable du problème original doit être dans au moins un sous problème
  - Chaque contrainte doit être dans au moins un sous problème
  - Si une variable apparaît plus d'une fois, les sous problèmes doivent être collés



- $O(nd^{w+1})$ où w est la grandeur du plus grand sous problème

Conclusion

- CSP = variables + domaines + contraintes
- Plusieurs problèmes peuvent être modélisés par le CSP
- Le *backtracking* est une approche de résolution largement utilisée
- Des heuristiques permettent de bien sélectionner l'ordre d'instantiation des variables des valeurs.
- Le *forward checking* et la consistance d'arcs pour la réduction de domaines réduise le facteur de branchement.
- Les approches locales (dont *min-conflicts*) sont très efficaces.
- La structure d'un graphe peut être exploitée pour réduire la complexité d'un problème.