Aix*Marseille université

État de l'art sur les principales heuristiques de choix de variables utilisées par les solveurs COP ou VCSP

Réalisé par Nassim MESRATI
Dirigé par Mr Cyril Terrioux

Articles à présenter

- Frédéric Boussemart and Fred Hemery and Christophe Lecoutre and Lakhdar Sais 1 ‘Boosting systematic search by weighting constraints’
- Philippe Refalo ‘Impact-Based Search Strategies for Constraint Programming’
- Laurent Michel1 and Pascal Van Hentenryck2 ‘Activity-Based Search for Black-Box Constraint Programming Solvers’
- Djamal Habet and Cyril Terrioux ‘Conflict History Based Branching Heuristic for CSP Solving’

Plan

I. Programmation sous contraintes
II. Résolution
III. Heuristiques de branchement
IV. Utilisation des heuristiques dans les solveurs
V. Conclusion

I.Programmation sous contraintes

Problèmes de Satisfaction de Contraintes (CSP)

Contraintes dans la vie réelle

- Affecter des stages à des étudiants
- Planifier le trafic aérien pour que tous les avions puissent décoller et atterrir sans se percuter

Problèmes de Satisfaction de Contraintes (CSP)

Instance $\mathcal{P} = (X, D, C)$ avec :

- $X = \{x_1, \ldots, x_n\}$ un ensemble de n variables,
- $D = \{d_{x_1}, \ldots, d_{x_n}\}$ un ensemble de domaines finis de taille au plus d
- C un ensemble de m contrainte $c = (S(c), R(c))$:
  - $S(c) \subseteq X$: portée
  - $R(c) \subseteq \prod_{x \in S(c)} d_x$: relation

Une solution du problème CSP: est une affectation **complète cohérente**.
Une affectation est dite complète si elle affecte une valeur à toutes variables
Une affectation est cohérente si elle respecte les contraintes du problème

Problèmes d'Optimisation sous Contraintes (COP)

Instance $\mathcal{P} = (X, D, C, f)$ avec :

- ➢ un CSP classique (X, D, C)
- ➢ f une fonction objective impliquant les variables $x_{i_1}, \ldots, x_{i_m}$

Une Solution du problème COP: est une affectation complète cohérente optimisant la fonction objective.

Problèmes de Satisfaction de Contraintes Valués (VCSP)

Instance $\mathcal{P} = (X, D, C, SV, \phi)$

- Un CSP classique (X, D, C)
- Une structure de valuation $SV = (E, \preceq, \oplus, \bot, \top)$
  - E : ensemble de valuation
  - R : relation d'ordre total
  - $\bigoplus$: loi de composition interne
  - $\bot$: élément minimum
  - T : élément maximum

II.Résolution

Problématique

L'espace de recherche est de taille $O(d^n)$ où :

d : est la taille du domaine des variables

n : le nombre de variables

L’augmentation de ces variables peut causer une explosion combinatoire dans un temps court, ce qui rend la résolution difficile et très coûteuse.

BackTracking

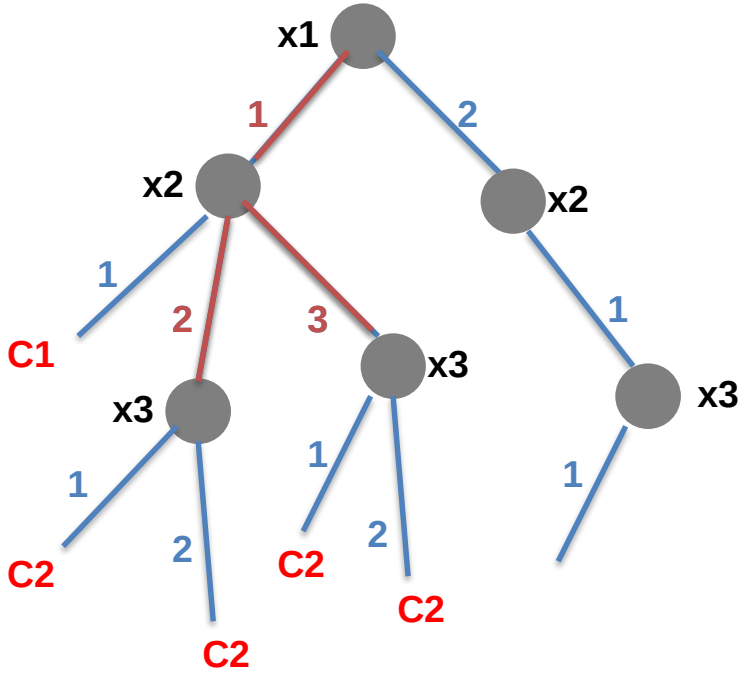
X = {x1,x2,x3 }

Dx1 = Dx2 = { 1, 2, 3 }

Dx3 = {1,2}

C1 = { x1 ≠ x2}

C2 = { x1 > x3 }



Backtracking + Filtrage

X = {x1,x2,x3 }

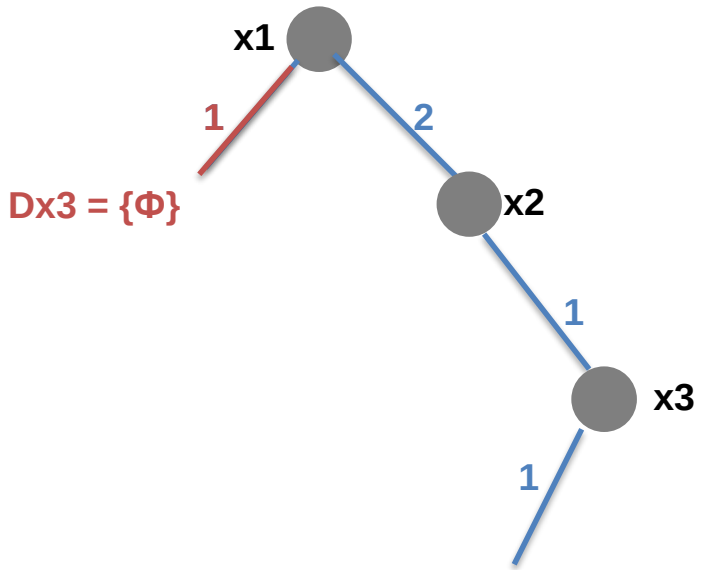
Dx1 = {1,2,3}

Dx2 = { ~~1~~, ~~2~~, 3 }

Dx3 = {~~1~~,~~2~~}

C1 = { x1 ≠ x2}

C2 = { x1 > x3 }



Résolution du cop en exploitant les outils CSP

- **Méthodes complètes:**
  - Développer des méthodes ad-hoc
  - Résoudre une succession d'instances CSP

- **Méthodes incomplètes**

Résolution du vcsp

- **Méthodes complètes:**
  - Approches énumératives (Branch and Bound)
  - Approches par programmation dynamique
  - Approches des poupées russes
- **Méthodes incomplètes**

Heuristiques de branchement

- CSP peut se faire en examinant un très grand nombre de combinaison.
- Une heuristique a pour objectif de guider la recherche.
- Permet de réduire la combinatoire et de guider la recherche vers les bonnes combinaisons.
- Le principe de « First fail » (l’echec d'abord) qui cherche à faire apparaître le plus rapidement possible des violations de contraintes

Redémarrage

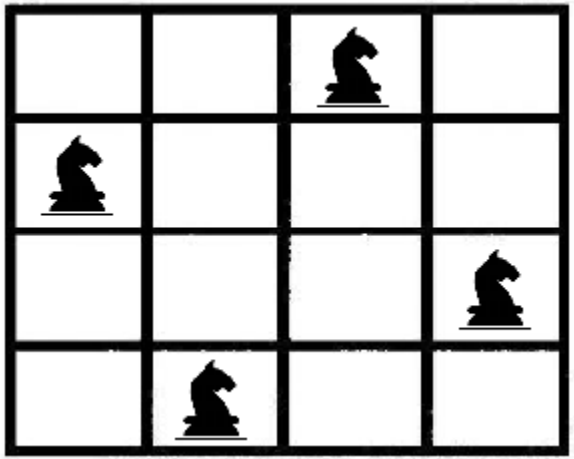
- Certaines exécutions d'un solveur complet et randomisé s'arrêtaient très rapidement, tandis que d'autres s'exécutent sur la même instance ont pris très longtemps.

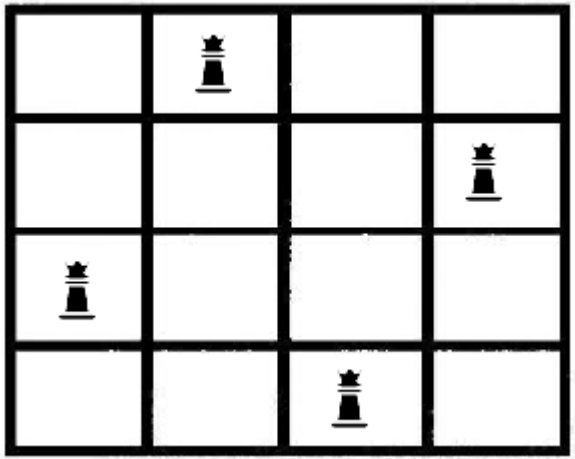
- L'algorithme de recherche / optimisation est interrompu au milieu d'une exécution puis redémarré de sorte qu'une partie différente de l'espace de recherche soit explorée.

l'heuristique wdeg

- L’une des heuristiques les plus utilisées.
- Simple à calculer.
- Basée sur la dureté de la contrainte.
- Associe un poids à chaque contrainte, avec un compteur qui calcule combien de fois cette contrainte nous a conduit à un échec.

l'heuristique wdeg





l'heuristique wdeg

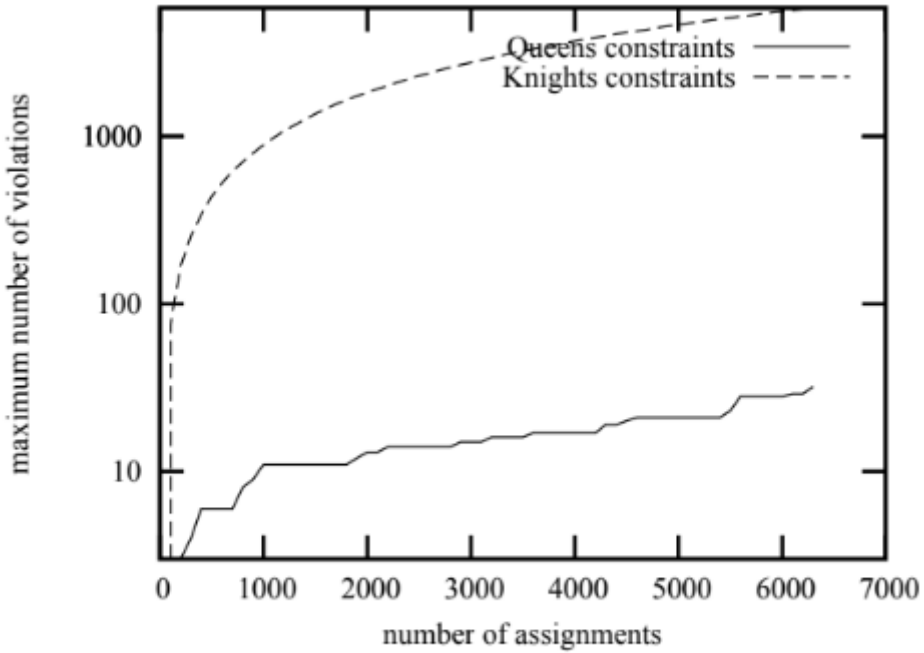


Figure 1. Evolution of constraint violations: 5 knights and 8 queens

l'heuristique wdeg

Algorithm 1 revise(C : Constraint, X : Variable) : boolean

1: **for** each $a \in dom(X)$ **do**
2: **if** $seekSupport(C, X, a) = false$ **then**
3: remove a from $dom(X)$
4: **if** $dom(X) = \emptyset$ **then**
5: $weight[C]$ ++
6: return $Dom(X) \neq \emptyset$

$$\alpha_{wdeg}(X_i) = \sum_{C \in \mathscr{C}} weight[C] \mid vars(C) \ni X_i \wedge \mid FutVars(C) \mid > 1$$

$FutVars(C)$ est le nombre des variables non instanciées

$vars(C)$ les supports de la contrainte c

l'heuristique wdeg

n	$instances$		$wdeg$	$\frac{dom}{wdeg}$	$\frac{dom}{ddeg}$
8	K_5	cpu	0.13	0.11	0.1
		#ccks	0.089M	0.083M	0.076M
		#asgs	0.14K	0.09K	0.06K
	$K_5 \oplus Q_8$	cpu	0.33	0.35	3.91
		#ccks	0.395M	0.39M	6.992M
		#asgs	0.5K	0.3K	6.3K
	$K_5 \otimes Q_8$	cpu	0.18	0.62	3.31
		#ccks	0.129M	0.807M	5.376M
		#asgs	0.1K	0.7K	5.7K
12	K_5	cpu	0.35	0.32	0.28
		#ccks	0.436M	0.413M	0.376M
		#asgs	0.3K	0.2K	0.1K
	$K_5 \oplus Q_{12}$	cpu	1.9	4.23	2,845.18
		#ccks	3.017M	7.613M	5,381.667M
		#asgs	1.8K	3.2K	2,174.2K
	$K_5 \otimes Q_{12}$	cpu	0.61	7.89	2,515.13
		#ccks	0.617M	12.869M	4,521.207M
		#asgs	0.3K	5.5K	2,017.2K

Table 1. Knights-Queens instances

Impact-Based Search (IBS)

- Inspirée des techniques d'**integer programming**
- Basées sur le concept d'**impact** d'une variable
- L'impact mesure l'importance d'une affectation de valeur à une variable pour la réduction de l'espace de recherche
- Le nombre des combinaisons possibles est une estimation:

$$P = |D_{x_1}| \times \ldots \times |D_{x_n}|$$

- L'affectation d'une valeur à une variable diminue la taille du domaine des autres variables après la propagation des contraintes

Impact-Based Search (IBS)

Impact d'une affectation:

$$I(x_i = a) = 1 - \frac{P_{after}}{P_{before}}$$

- Problème: l'impact de chaque valeur à chaque variable non instanciée ce qui cause une surcharge

- Des expériences ont montré que l'impact ne change pas trop d'un noeud à un autre

Impact-Based Search (IBS)

l'impact d'une affectation sur un nœud donné peut être la moyenne :

$$\bar{I}(x_i = a) = \frac{\sum_{k \in K} I^k(x_i = a)}{|K|}$$

Avec

K : l'ensemble des index observés

$\bar{I}$: la moyenne d'impact

Impact-Based Search (IBS)

l'impact d'une variable:

- L'impact d'une variable peut être la moyenne des impacts $\bar{I}(x_i = a)$ avec $a \in D_{x_i}$

$$\tilde{I}(x_i) = \frac{\sum_{a \in D'_{x_i}} \bar{I}(x_i = a)}{|D'_{x_i}|}$$

- Approche peu précise

- Le but est de choisir une variable ayant le plus grand impact lors de l'affectation d'une des valeurs restantes dans son domaine.

Impact-Based Search (IBS)

l'impact d'une variable:

- En tenant compte de l'impact sur tous les domaines des variables:
- L'estimation de la taille de recherche avec une affectation sur une variable x_i est

$$P \times (1 - \bar{I}(x_i = a))$$

- En considérant toutes les affectations possibles sur x_i:

$$\mathcal{I}(x_i) = \sum_{a \in D'_{x_i}} 1 - \bar{I}(x_i = a)$$

Impact-Based Search (IBS)

Redémarrage:

- Imposer un nombre maximum d'échecs de **3n** où **n** est le nombre de variables, cette valeur s'appelle **the cutoff**

- A chaque redémarrage, nous augmentons cette valeur en la multipliant par une valeur e> 1

- L'augmentation du seuil nous permet de garantir que le processus de recherche est complet.

- Meilleurs résultats ont été obtenus avec e = $\sqrt{2}$

Activity-Based Search (ABS)

- Méthode basée sur les techniques de filtrage qui suppriment les valeurs inutiles
- Elle exploite ces informations et les conserve
- Les résultats expérimentaux sur une variété de points de repère montrent que la recherche par activité est plus robuste que les autres méthodes heuristiques et peut produire des améliorations significatives des performances.

Activity-Based Search (ABS)

Soit P = {X, D, C} un problème sous forme CSP

- Apres l'application du filtrage et la propagation des contraintes, on obtient un nouveau domaine :

$$D' \subseteq D$$

- Et un nouveau sous ensemble des variables instanciées :

$$X' \subseteq X$$

Tel que :

$$\begin{aligned} &\forall x \in X' &&: D'(x) \subset D(x); \\ &\forall x \in X \setminus X' &&: D'(x) = D(x). \end{aligned}$$

Activity-Based Search (ABS)

On note A(x) l'activité de la variable x

- A(x) est mise à jour à chaque nœud k de l'arbre de recherche indépendamment du résultat (succès ou échec) par les deux règles suivantes:

$$\begin{aligned} \forall x \in X \text{ s.t. } |D(x)| > 1 &: A(x) = A(x) \cdot \gamma \\ \forall x \in X' &: A(x) = A(x) + 1 \end{aligned}$$

- Avec γ paramètre de dégradation où 0 <= γ <= 1. L'activité d'une affectation x = a à un nœud de recherche k est calculée comme suit:

$$A_k(x = a) = |X'|.$$

Activity-Based Search (ABS)

- Comme dans IBS, l'activité d'une affectation est peut être estimée par une moyenne:

$$\tilde{A}(x=a) = \frac{\sum_{k \in \mathcal{K}} A_k(x=a)}{|\mathcal{K}|}$$

- Il est plus simple de privilégier une somme pondérée:

$$\tilde{A}_1(x=a) = \frac{\tilde{A}_0(x=a) \cdot (\alpha - 1) + A_k(x=a)}{\alpha}$$

Activity-Based Search (ABS)

- Abs sélectionne la variable x avec le ratio $A(x)/|D(x)|$
- En cas d'égalité, on tire une variable aléatoire
- Abs sélectionne la valeur **a** avec l'activité minimale :

$$\text{argMin}_{v \in D(x)} \tilde{A}(x = v)$$

Activity-Based Search (ABS)

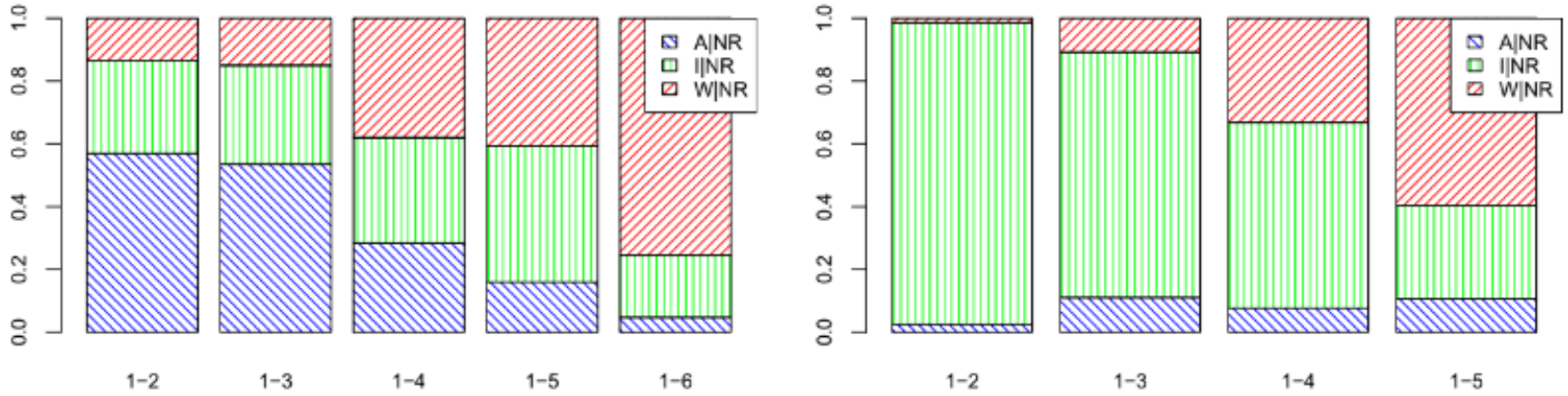


Fig. 1. Knapsack, no-restart, decision variant (left) and Optimization variant (right)

Conflict History Based Branching Heuristic

- Basée sur l'historique des échecs de recherche qui se produisent suite à des propagations des contraintes.

- Récompenser les scores des contraintes récemment impliquées dans les conflits à l'aide de la technique moyenne pondérée de récence exponentielle issue de l'apprentissage par renforcement.

Conflict History Based Branching Heuristic

Exponential Recency Weighted Average :

- Résoudre le problème des bandits afin d'estimer la récompense attendue de différentes actions.

- En donnant aux données récentes des poids plus élevés que les plus anciens.

Définie comme suit:

$$\bar{y}_m = \sum_{i=1}^{m} \alpha.(1-\alpha)^{m-i}.y_i \qquad 0 < \alpha < 1$$

Conflict History Based Branching Heuristic

- Pour chaque contrainte c_j , on maintient un score $q(c_j)$ initialisé à 0 au début de la recherche.

- Si c_j mène vers un échec après le filtrage et que le domaine soit vide alors $q(c_j)$ est mis à jour par la formule suivante:

$$q(c_j) = (1 - \alpha) \times q(c_j) + \alpha \times r(c_j)$$

- $r(c_j)$ est la récompense basée sur la façon dont c_j s'est récemment produit dans des conflits

Conflict History Based Branching Heuristic

$$r(c_j) = \frac{1}{\#Conflicts - Conflict(c_j) + 1}$$

$\#Conflicts$ est le nombre de conflits survenus depuis le début de la recherche.

- Le score **Conflict-History** d'une variable est calculé comme suit:

$$chv(x_i) = \frac{\sum_{c_j \in C | x_i \in S(c_j) \wedge |Uvars(c_j)| > 1} q(c_j)}{|D_i|}$$

Conflict History Based Branching Heuristic

- Au début de la recherche, toutes les variables ont le même score égal à 0
- Pour éviter un choix aléatoire, la formule chv est calculée comme suit:

$$chv(x_i) = \frac{\sum_{c_j \in C | x_i \in S(c_j) \wedge |Uvars(c_j)| > 1} (q(c_j) + \delta)}{|D_i|}$$

Avec δ un nombre real positif proche de 0

Conflict History Based Branching Heuristic

Redémarrage:

- Les conflits ne sont pas réinitialisés pendant le redémarrage

- Certains poids des contraintes peuvent être inchangés pour plusieurs étapes de recherche.

- Ces contraintes peuvent avoir des scores élevés alors que leur importance ne semble pas élevée pour la partie actuelle de la recherche.

Pour éviter cette situation, CHS propose de lisser les scores q (cj)

Conflict History Based Branching Heuristic

Redémarrage:

Pour toute contrainte $c_j \in C$ à chaque redémarrage :

$$q(c_j) = q(c_j) \times 0.995^{\#Conflicts - Conflict(c_j)}$$

IV.L’utilisation des heuristiques dans les solveurs modernes

XCSP3

- Est un format basé sur XML
- Conçu pour représenter des instances de problèmes à contraintes combinatoires sous l'angle de la programmation par contraintes (CP)

BTD

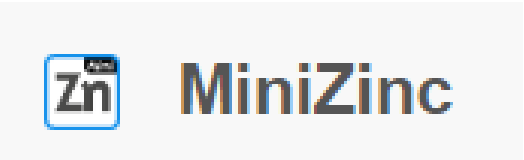
- Il utilise « tree-decompositions » grâce à l'heuristique **H5-TD-WT**
- Choix des variables dans **les clusters**: **dom/wdeg**
- Choix de valeur: **lexico**
- Redémarrage **: geometric restart** basé sur le nombre des backtrack

Concrete 3.9.2

- Choix des variables: **dom/wdeg** avec **« Incremental computation»**
- L'égalité est cassée par un choix aléatoire
- Choix de valeur:
  - the best known value first par défaut
  - BBS (Black-Box Search)

MiniZinc Challenge

- Un concours annuel de solveurs de programmation par contraintes
- Il traduit les modèles de contraintes en **FlatZinc**



iZplus

- Les échecs survenus pendant la recherche sont enregistrés
- L'ordre de sélection des variables est modifié à l'aide de ces informations
- La taille du domaine et le nombre des contraintes pondérées sont également utilisés.

Conclusion

- Une heuristique permet de guider la recherche pour réduire le temps de résolution
- Le choix d'une meilleure heuristique dépend de la taille et de l'instance à résoudre