Structure de problème CSP Structuré

H.BELLEILI

Position du problème

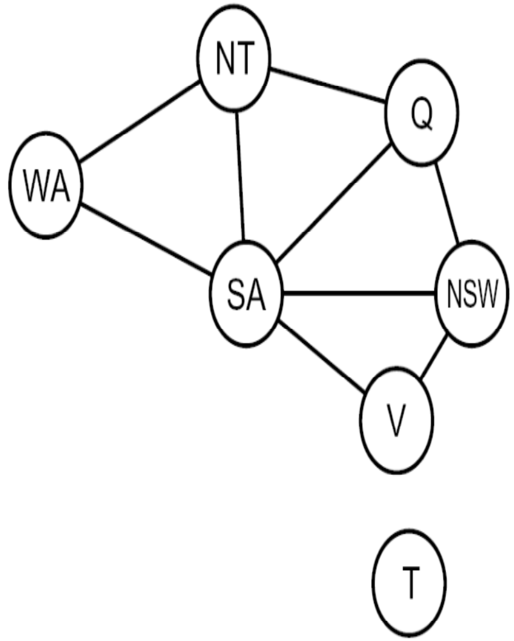
Å Soit un graphe de contraintes ayant « n » nî uds et chaque nî ud a un domaine discret de « d » valeurs.

Å Quel sera le coût de la solution au pire cas?

Structure du problème

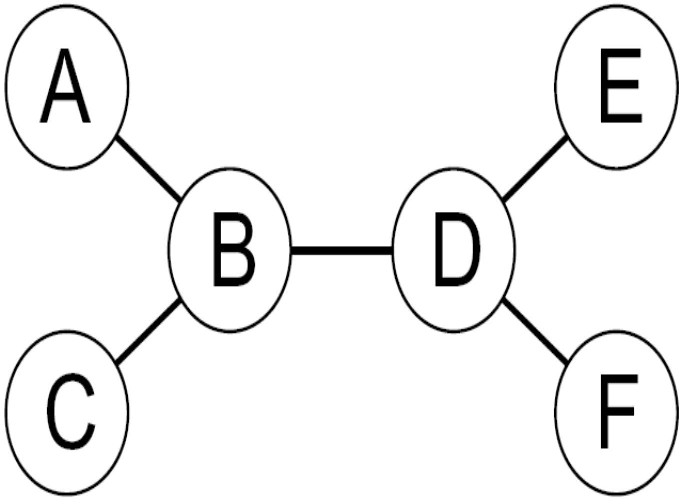
- Si lôon pouvait diviser le problème en sous problèmes indépendants,

- Les sous poblèmes indépendants sont identifiables comme des composants connectés du graphe de contraintes:
  - ***Diviser et reigner!***



- Soit un graphe de n variables qui peut être divisé en sous problèmes de seulement c variables chacun:
  - Quel serait le Coût de la solution au pire cas?
  - $O((n/c)(d^c))$, ***linéaire en n*** ,
  - ex. n = 80, d = 2, c =20, vitesse de la recherche 10 million noeuds/sec
  - Original problem: 2^{80} = **4 billion years**
  - Four subproblems: 4 x 2^{20} = **0.4 seconds**

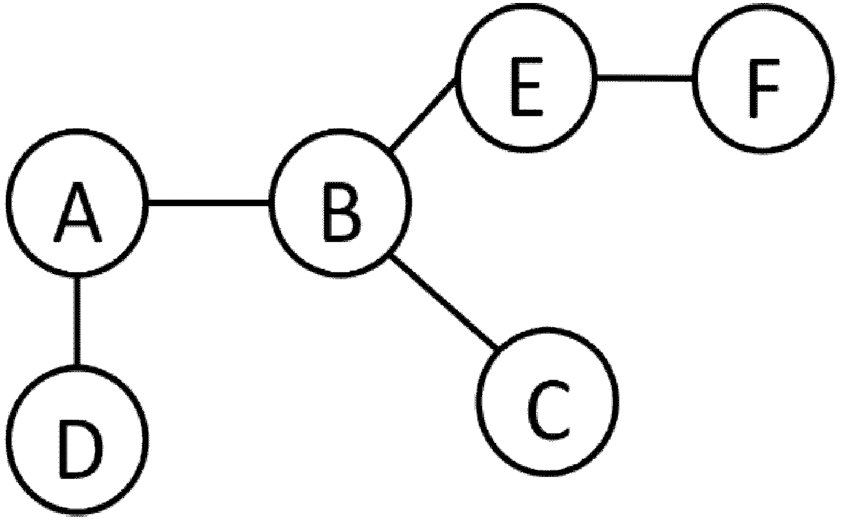
CSPs structurés en arbre

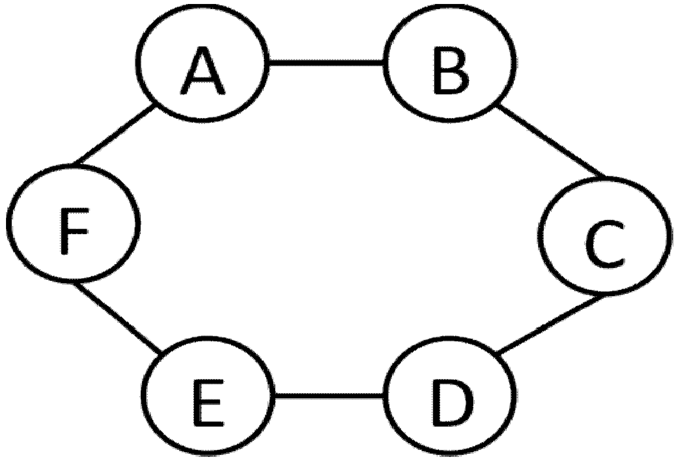


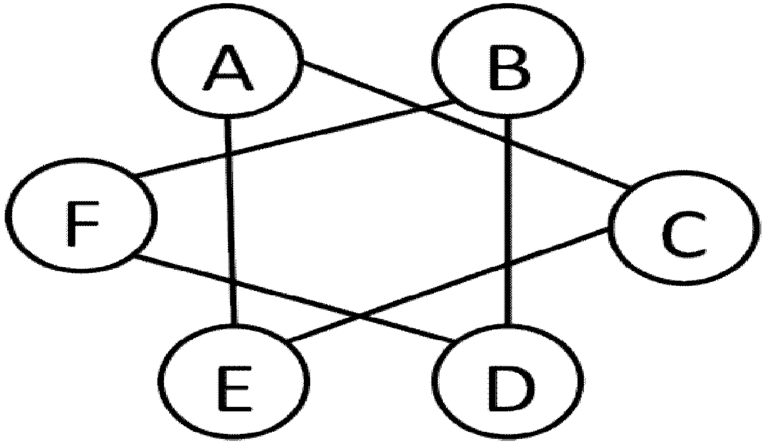
- Theoreme: si le graphe de contraintes n'a pas de boucles, le CSP peut être résolu en un temps linéaire en n et quadratique en la taille du domaine d $O(n\ d^2)$
  - À Comparer avec les CSPs généraux, où le temps au pire cas est $O(d^n)$

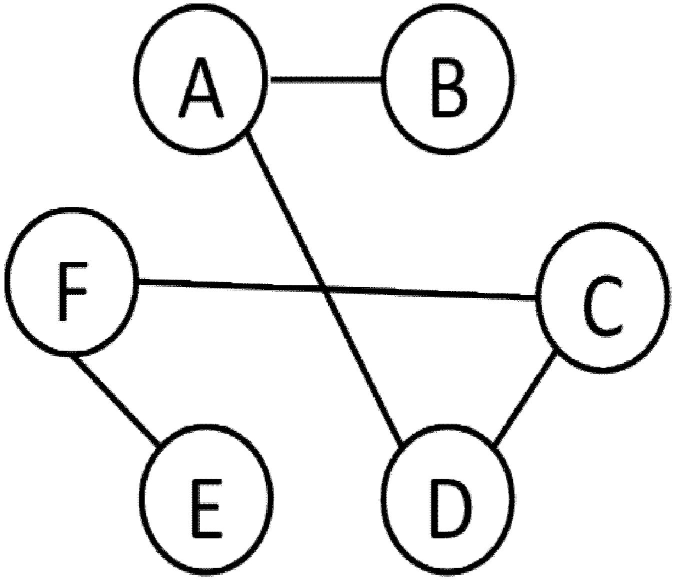
Tree-Structured CSPs

- Donc le problème de CSP structuré en arbre est beaucoup plus efficace quôun problème non structuré.
- Il serait intéressant de trouver une structure en graphe dôun CSP ou à défaut une structure proche dôun arbre.


E
F
A
B
D
C


A
B
F
C
E
D

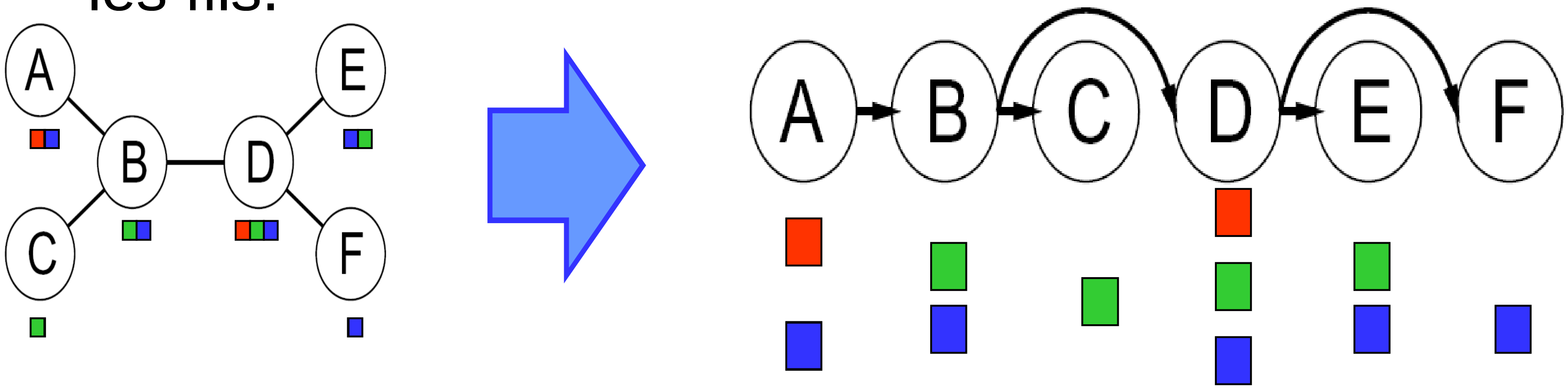

A
B
F
C
E
D


A
B
F
C
E
D

CSPs structurés en arbre

Å Algorithme pour les CSPs structurés en arbre:

ï Ordre: choisir une variable racine. Ordonner les variables de manière à ce que les parents précèdent les fils.



ï Supprimer en **chainage arrière** : For i = n : 2, appliquer removeInconsistent(Parent(X_i),X_i)

ï Affectation en **chainage avant**: For i = 1 : n, assign X_i consistently with Parent(X_i)

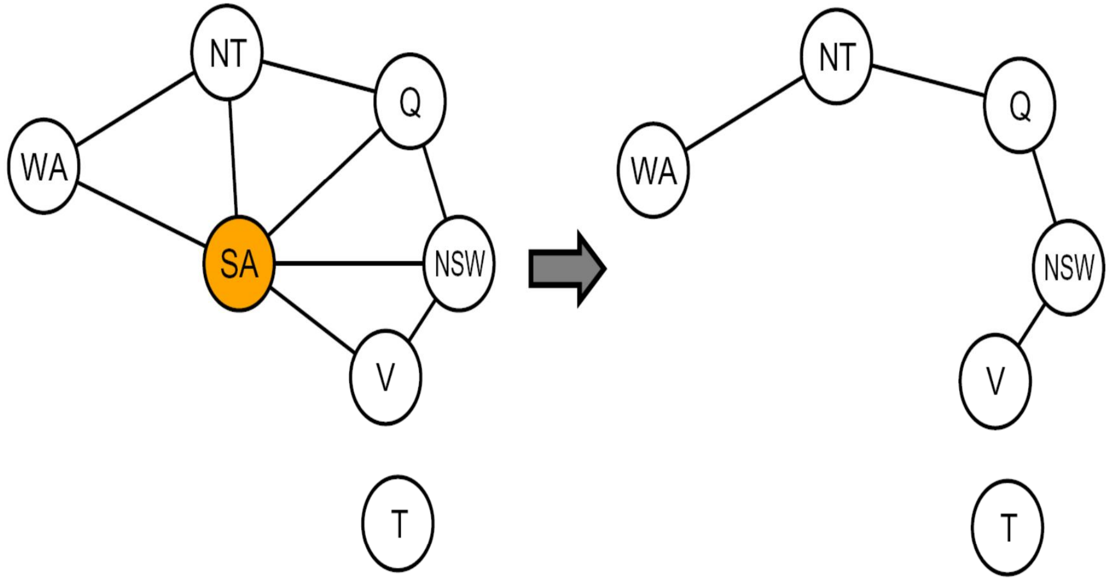
Å Runtime:

$O(n.d^2)$ (pourquoi?)

RunTime

- Dans un arbre, chaque nî ud a un seul parent au plus, donc le nombre dôarc est majoré par n
- Jôinspecte chaque arc vers la gauche puis vers la droite donc côest de lôordre de n puisque je visite chaque arc un nombre de fois constant.
- À chaque arc je regarde toutes les paires de valeurs du domaine dans la tête et dans la queue de lôarc et je vérifie la consistance ceci est d au carré
- Le runtime est $O(n.d^2)$

Nearly Tree-Structured CSPs



- *Conditioning*: instancier une variable, élaguer les domaines de ces voisins
- *Cutset conditioning*: (dit coupe cycle) instancier (en considérant tous les cas) lʼensemble de variables de manière à ce que le reste du graphe de contraintes est un arbre
- Si lʼensemble coupe cycle a une taille c (variables) alors lʼexécution totale est $O((d^c)(n\text{-}c)d^2)$, très rapide pour c petit
- Exemple pour 80 variables, c=10, le temps dʼexécution est 0.029 seconds au lieu de 4 billion years.

Suite

- Si le graphe de contrainte est un « quasi arbre » alors c sera petit et les économies par rapport au backtracking simple seront considérables.
- Dans le pire cas c peut valoir jusqu'à (n-2) alors c très grand.
- La détermination du plus petit ensemble coupe-cycle est un problème NP-difficile,
- Il existe plusieurs algorithmes efficaces qui permettent d'obtenir une approximation.

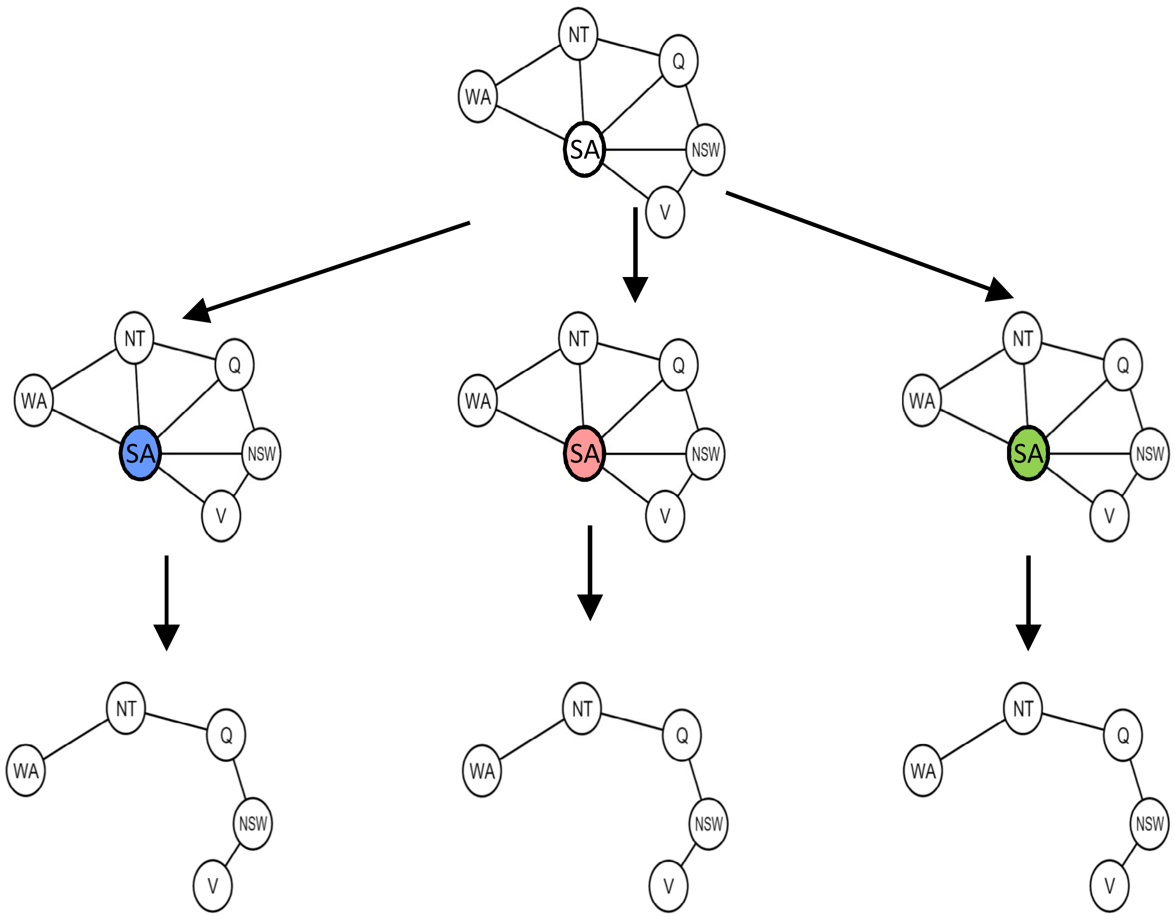
Cutset Conditioning

Choisir un cutset

Instantier le cutset (tous les cas possibles)

pour chaque affectation calculer les CSP résiduel

Résoudre chaque CSP (sous forme d'arbre)



Summary: CSPs (from [..?])

- CSPs are a special kind of search problem:
  - States are (partial) assignments of values to variables
  - Goal test defined by constraints
  - Formal language => ***general*** algorithms and heuristics
- Basic algorithm: backtracking search
- Speed-ups:
  - Ordering
  - Filtering
  - Structure