Systèmes Multi-Agent
Satisfaction et Optimisation de contraintes distribuées

Emmanuel ADAM

Université Polytechnique des Hauts-De-France

UPHF/INSA HdF

Rappel sur le CSP

Définition d'un Problème de Satisfaction de Contraintes

- Soit X un ensemble de variables x_i,
- Soit D le domaine de valeurs des variables de X,
- Soit C un ensemble de contraintes sur un ensemble de variables x_i
- Résoudre un CSP consiste à affecter des valeurs aux variables de X respectant les contraintes de C

Exemple de Problème de Satisfaction de Contraintes

- Soient $X = \{x_0, x_1, x_2, x_3\}$ et $D = \{0, 1\}$
- Soit $C = \{(x_0 \neq x_1), (x_0 \neq x_3), (x_1 \neq x_2), (x_2 \neq x_3)\}$
- $\{(x_0 \leftarrow 0, x_1 \leftarrow 1, x_2 \leftarrow 0, x_3 \leftarrow 1\}$ est une solution

Exemple d'algorithme avec Retour Arrière (BackTrack)

Exemple d'adaptation de l'algo de recherche de solutions avec retour arrière présenté en Master 1 TNSI :

procedure $\textsc{CentralizedBT}(i, \mathit{affectations})$
    if $i > n$ **then** return *affectations*
    else
        $\mathit{acceptableValues} \leftarrow \textsc{CoherentValues}(\mathrm{C}, \mathrm{affectations})$ ▷ *acceptableValues is a set of values that do not violate constraints in C, given affectations already done*
        for all $x \in \mathit{acceptableValues}$ **do**
            $\mathit{newAffectation} \leftarrow \textsc{CentralizedBT}(i+1, \mathit{affectations} \cup (x_i \leftarrow x))$
            if $\mathit{newAffectation} \neq \emptyset$ **then** return *affectations*
            end if
        end for
        return $\emptyset$
    end if
end procedure

Distribution d'un Problème de Satisfaction de Contraintes

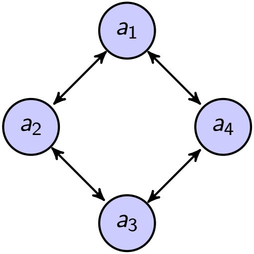
Distributed Constraint Solving Problem : DCSP

- Soit un CSP définit par X, D, C
- La distribution d'un CSP consiste à définir un agent a_i par variable x_i.
- Chaque agent a_i a la responsabilité de l'affectation d'une valeur à sa variable x_i, en fonction des valeurs définies par les agents de son voisinages, obtenues par communication

Distribution d'un Problème de Satisfaction de Contraintes

Exemple de DCSP : coloration de graphe

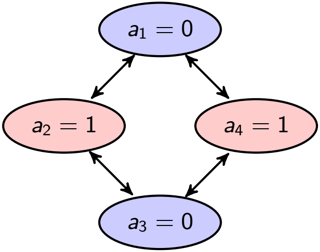
- Soit un graphe constitué de 4 noeuds représentant chacun une valeur x_i
- Soit $D = \{0, 1\}$ le domaine de valeurs de $\{x_1, x_2, x_3, x_4\}$
- Les contraintes sont qu'aucun noeud ne peut avoir la même valeur que son voisin. Donc ici, $C = \{(x_1 \neq x_2), (x_1 \neq x_4), (x_2 \neq x_3), (x_3 \neq x_4)\}$
- on définit alors 4 agents a_1, a_2, a_3, a_4 responsables respectivement de x_1, x_2, x_3, x_4



Distribution d'un Problème de Satisfaction de Contraintes

Exemple de DCSP : coloration de graphe

- Soit un graphe constitué de 4 noeuds représentant chacun une valeur x_i
- Soit $D = \{0, 1\}$ le domaine de valeurs de $\{x_1, x_2, x_3, x_4\}$
- Les contraintes sont qu'aucun noeud ne peut avoir la même valeur que son voisin. Donc ici, $C = \{(x_1 \neq x_2), (x_1 \neq x_4), (x_2 \neq x_3), (x_3 \neq x_4)\}$
- on définit alors 4 agents a_1, a_2, a_3, a_4 responsables respectivement de x_1, x_2, x_3, x_4



Principe de résolution

Agents coopératifs

- Chaque agent a_i “connait” ses voisins :
  - il connaît leurs valeurs (si définies),
  - et peut connaître leurs contraintes (partiellement ou non).
- Plusieurs versions :
  - **Algorithme de filtrage** :
    Si une valeur de a_i empêche son voisin a_j de trouver une valeur, celui-ci l'en informe : il restreint le domaine de valeurs de a_i
  - **BackTracking Décentralisé (Asynchronous BackTracking, ...)** :
    Si un agent a_i est dans l'incapacité de trouver une valeur suite aux valeurs de ses voisins, il calcule l'ensemble des valeurs interdites (les *nogoods*) qu'il leurs envoie.
    Possibilité à deux agents non liés de se connecter pour accélérer la résolution.

ABT : algo par recherche arrière décentralisé

Agents coopératifs

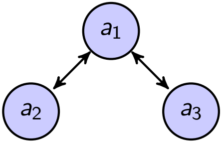
- Connaissance d'un agent a_i :
  - *neighborgs* : liste de couples $(voisin, valeur)$,
  - *noGoods* : liste de 'valeurs interdites', à éviter par ses voisins, que a_i a calculé pour qu'il puisse affecter une valeur à x_i
  - *constraints* : liste de contraintes que l'agent doit respecter
- Fonctionnement :
  - Chaque agent a_i initialise sa variable x_i en fonction du domaine D et de *constraints*
  - Chaque agent informe ses voisins du choix de x_i
  - Chaque agent vérifie si sa valeur est toujours consistante avec les contraintes et les nouvelles valeurs reçues.
    Si aucune valeur de D ne peut être affectée à x_i, l'agent a_i demande un *backtrack* au voisinage, si possible.

ABT : algo par recherche arrière décentralisé

Agents coopératifs

- Fonctionnement :
  - Le backtrack demandé au voisinage consiste pour l'agent a_i à calculer les $\boxed{\textit{noGoods}}$, valeurs que les voisins ne peuvent avoir pour libérer l'inconsistance.
    - Si aucun *noGood* n'est trouvé, la backtracking est impossible, la réolution échoue.
    - Chaque *noGood* calculé est envoyé à l'agent impliqué de priorité la plus basse (le moins contraint)
  - Lorsqu'un agent reçoit un *noGood*, si celui-ci implique un autre agent qu'il ne connaît pas, il l'ajoute à son voisinage *neighborgs*.
    Puis il tente de trouver une nouvelle valeur qu'il transmet à l'agent émetteur du *noGood*
- L'algo prend fin lorsque les agents n'émettent plus de message ou qu'ils ont transmis un message d'echec.

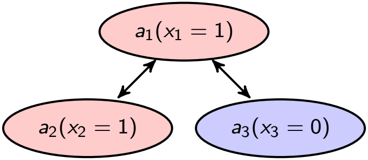
ABT : Exemple



Exemple de calcul de nogoods

Soit 3 agents a_1, a_2 et a_3, dont les variables x_1, x_2 et x_3 prennent leurs valeurs dans $\{0, 1\}$ en respectant les contraintes $(x_1 \neq x_2)$ et $(x_1 \neq x_3)$

ABT : Exemple

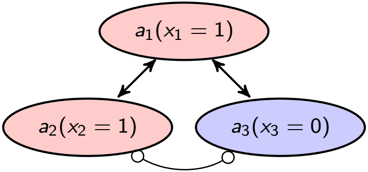


Exemple de calcul de nogoods

Soit 3 agents a_1, a_2 et a_3, dont les variables x_1, x_2 et x_3 prennent leurs valeurs dans $\{0, 1\}$ en respectant les contraintes $(x_1 \neq x_2)$ et $(x_1 \neq x_3)$

1. Les agents choisissent une valeur : $a_1(x_1 \leftarrow 1)$, $a_3(x_3 \leftarrow 0)$ et $a_2(x_2 \leftarrow 1)$ et en informe les autres
2. a_1 reçoit donc "$a_2(x_2 \leftarrow 1)$", teste la cohérence et détecte un problème
3. a_1 n'a pas d'autres valeurs libres pour x_1 sans violer de contrainte
4. a_1 calcule les *noGoods* (valeurs à éviter) par rapport à sa valeur $x_1 = 1$: il trouve $\{(x_2 = 1), (x_2 = 0 \wedge x_3 = 1)\}$
5. a_1 envoie chaque *noGood* à l'agent adéquat (de plus basse priorité, ou conflictuel), ici a_2, et supprime de ses connaissance l'affectation $(x_2 = 1)$

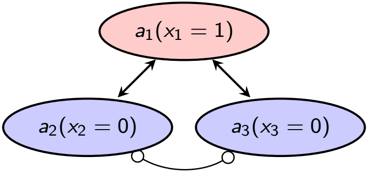
ABT : Exemple



Exemple de calcul de nogoods

6. a_2 reçoit "$(x_2 = 0 \wedge x_3 = 1)\}$"
   a_2 ajoute a_3 à sa liste de voisin. Il sait alors que $(x_1 = 1)$ et $(x_3 = 0)$
   a_3 s'ajoute à la liste des voisins de x_3

ABT : Exemple



Exemple de calcul de nogoods

7. a_2 cherche une valeur respectant les contraintes et ne se trouvant pas dans les *noGoods*.
   a_2 trouve $(x_2 \leftarrow 0)$ et communique cette information à ses voisins
   (Remarque : le nogood n'est pas utile dans cet exemple pour la résolution)
8. a_1 et a_3 reçoivent cette information cohérente avec leurs vues et ne changent pas leurs valeurs
9. a_2 traite l'autre *noGood* : "$(x_2 = 1)$"
   Ce *noGood* n'est pas en conflit avec la nouvelle valeur de x_2 ; a_2 reste sur $(x_2 = 0)$
10. Plus aucun message n'est échangé, la solution est trouvée

Exemple d'algorithmes pour ABT

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal,March 1, 2010
Pour un agent a_i donné :

procedure $\textsc{receiveAffectation}(a_j, x_j)$
  $localView \leftarrow localView + (a_j, x_j)$
  $\textsc{checkLocalView}$
end procedure

procedure $\textsc{checkLocalView}$
  if (not $\textsc{consistant}(localView, x_i)$) **then**
    $x \leftarrow \textsc{findConsistantValue}(localView)$
    if $(x = \emptyset)$ **then** $\textsc{BackTrack}$
    else
      $x_i \leftarrow x$
      $\forall a_k \in neighborgs_{a_i}$, $\textsc{ask}(a_k,$ "receiveAffectation(a_i, x_i)")
    end if
  end if
end procedure

Exemple d'algorithmes pour ABT

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal,March 1, 2010

procedure BACKTRACK
  $noGoods \leftarrow$ FINDNOGOODS($localView$)
  if ($\emptyset \in noGoods$) **then**
    BROADCAST("No solution ! !")
    EXIT
  else
    for all ($ng \in noGoods$) **do**
      select $(a_j, x_j) \in ng | hasMinPriority(a_j)$
      ASK(a_j, "receiveNoGood(a_i, ng)")
      $localView \leftarrow localView - (a_j, x_j)$
    end for
  end if
  CHECKLOCALVIEW
end procedure

Exemple d'algorithmes pour ABT

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal,March 1, 2010

procedure RECEIVENOGOOD(a_j, ng)
  STOREASCONSTRAINT(ng)
  for all $(a_k \in ng | ng \notin neighborgs_{a_i})$ **do**
    ASK(a_k,"addNeighborg(a_i)")
    $neighborgs \leftarrow neighborgs \cup \{a_k\}$
    $localView \leftarrow localView \cup \{(a_k, x_k)\}$
  end for
  $oldX \leftarrow x_i$
  CHECKLOCALVIEW
  if $(oldX \neq x_i)$ **then** ASK(x_j,"receiveAffectation(a_i, x_i)")
  end if
end procedure

procedure ADDNEIGHBORG(a_j)
  $neighborgs \leftarrow neighborgs \cup \{a_j\}$
end procedure

ABT : avantages et inconvénients

Complétude

L'algorithme d'Asynchronous BactTracking (ABT) est complet :

- ABT trouvera la solution si elle existe,
- autrement, ABT conclura sur un échec en temps fini.

Problèmes

- Répartition de la charge de travail : la priorité des agents est fixe $\Longrightarrow$ le recalcule de valeurs est demandé aux mêmes agents.
- Heuristique de choix de la valeur x_i pour l'agent a_i ne prend pas en compte les contraintes des voisins $\Longrightarrow$ conflits

AWC : améliorations de ABT

Quelques améliorations à ABT

- Priorité dynamique des agents :
  - l'agent réalisant un BackTracking devient le plus prioritaire de son voisinage,
  - pour un agent a_i, le choix de la valeur x_i est réalisée en fonction des contraintes des agents de priorités inférieure à a_i

Complétude

L'algorithme d'Asynchronous Weak Commitment search est complet :

- AWC trouvera la solution si elle existe,
- autrement, AWC conclura sur un échec en temps fini.

Exemple d'algorithmes pour AWC

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal,March 1, 2010
Pour un agent a_i donné, reprise des algos de ABT sauf le fait que les agents stockent en plus de la valeur des voisins, leurs priorités.

procedure CHECKLOCALVIEW
  if (not CONSISTANT($localView$,x_i)) **then**
    $x \leftarrow$
FINDCONSISTANTVALUEMINIMISINGCONSTRAINTSLOWAGENTS($localView$)
    if ($x = \emptyset$) **then** BACKTRACK
    else
      $x_i \leftarrow x$
      $\forall a_k \in neighborgs_{a_i}$, ASK($a_k$, "receiveAffectation($a_i$,$x_i$)", $priority_{a_i}$)
    end if
  end if
end procedure

Exemple d'algorithmes pour ABT

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal,March 1, 2010

```
procedure BACKTRACK
    ng ← GENERATENOGOOD(localView)
    if (ng = ∅) then
        BROADCAST("No solution ! !")
        EXIT
    else
        if (ng is a new noGood) then
            for all ((a_j, x_j) ∈ ng) do
                ASK(a_j, "receiveNoGood(a_i, ng, priority_{a_i})")
            end for
            priority_{a_i} ← priority_{a_i} + max(neighborgs.priority)
            xi ←
FINDCONSISTANTVALUEMINIMISINGCONSTRAINTSLOWAGENTS(localView)
            ∀a_k ∈ neighborgs_{a_i}, ASK(a_k, "receiveAffectation(a_i,x_i)", priority_{a_i})
        end if
    end if
    CHECKLOCALVIEW
end procedure
```

Distributed BreakOut : principes

Méthode de descente (Hill Climbing)

1. Chaque agent a_i prend une valeur x_i aléatoirement
2. Chaque agent tente de diminuer le nombre de contraintes violées en choisissant une autre valeur
3. Le processus (2) s'arrête lorsque plus aucune amélioration n'est possible

Incomplétude

L'algorithme Distributed BreakOut n'est pas complet.

- Il peut mener à des quasi-minima-locaux :
  Aucun agent ne peut bouger sans empirer la situation de son voisinage.

Distributed BreakOut : principes

Tenter d'éviter les minimas locaux

- Chaque violation de contrainte a son propre poids.
- Une situation de minimum local apparaît pour un agent lorsqu'aucune modification de lui et de ses voisins ne peut diminuer le coût local de violations des contraintes.
- Idée : augmenter le poids, donc le coût (la punition) pour les contraintes non respectées.

1. Chaque contrainte a initialement un poids $= 1$.
2. Lorsqu'un agent est en situation de minimum local, il incrémente le poids des contraintes non respectées.
3. Le coût de non respect des contraintes est relatif à la somme de leurs poids.

Exemple d'algorithmes pour Distributed BreakOut

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal, March 1, 2010

Pour un agent a_i donné :

Begin-DEFINITION OF AN AGENT
  AgentAddress [] neighborgs ; ▷ *addresses of the neighbors*
  int [] valuesOfAgents ; ▷ *values of the local set of agents*
  int [] receivedImprovements ; ▷ *values of the possible improvements of the other agents*
  int [] constraintsWeights ; ▷ *weights of the constraints with the neighbors*
  int [] notRespectedConstraints ; ▷ *nb of violated constraints relative to the neighbors*
  int value ; ▷ *value of the agent*
  int bestValue ; ▷ *best possible next value of the agent*
  int maxImprovement ; ▷ *best possible improvement*
  int cost ; ▷ *cost due to the current non respected constraints*
  int totalOtherCosts ;▷ *sum of the other costs (if =0 and cost=0, then End of the search !)*
End

Exemple d'algorithmes pour Distributed BreakOut

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal,March 1, 2010
Pour un agent a_i donné :

procedure $\textsc{receiveAffectation}(a_j, x_j, weight_{a_j})$
    $\textsc{updateWeightConstraints}(weight_{a_j})$
        ▷ *constraintsWeights[j]=max(weight$_{a_j}$,constraintsWeights[j])*
    $valuesOfAgents \leftarrow localView + (a_j, x_j)$ ▷ *valuesOfAgents[j]=x_j*
    $allReceived \leftarrow \textsc{isGotAllAffectations}()$
    if $(allReceived)$ **then**
        $\textsc{sendImprovement}$
    end if
end procedure

Exemple d'algorithmes pour Distributed BreakOut

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal,March 1, 2010
Pour un agent a_i donné :

procedure SENDIMPROVEMENT
    $notRespectedConstraints \leftarrow$ COMPUTENBOFVIOLATIONS($valuesOfAgents$)
    $cost \leftarrow$ COMPUTECOSTS($notRespectedConstraints$, $constraintsWeights$)
    $(bestValue, maxImprovement) \leftarrow$
        COMPUTEBESTVALUE($valuesOfAgents$, $constraintsWeights$)
    for all $a_j \in neighborgs_{a_i}$ **do**
        ASK(a_j, "receiveImprovement(a_i, $maxImprovement$, $cost$)")
    end for
end procedure

Exemple d'algorithmes pour Distributed BreakOut

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal,March 1, 2010
Pour un agent a_i donné :

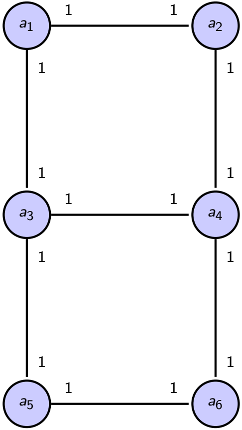
procedure $\textsc{ReceiveImprovement}(a_j, \mathit{improvement}_{a_j}, \mathit{cost}\,a_j)$
  $\mathit{receivedImprovement}[j] \leftarrow \mathit{improvement}_{a_j}$
  $\mathit{totalOtherCosts} \leftarrow \mathit{totalOtherCosts} + \mathit{cost}\,a_j$
  $\mathit{allReceived} \leftarrow \textsc{isGotAllImprovements}()$
  if $(\mathit{allReceived})$ **then**
    $\textsc{sendOk}$
  end if
end procedure

Exemple d'algorithmes pour Distributed BreakOut

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal,March 1, 2010
Pour un agent a_i donné :

procedure SENDOK
    $isMax \leftarrow false$
    $totalImprovement \leftarrow 0$
    for all $a_k \in neighborgs_{a_i}$ **do**
        $isMax \leftarrow isMax \land (maxImprovement \geqslant receivedImprovements[k])$
        $totalImprovement \leftarrow totalImprovement + receivedImprovements[k]$
    end for
    if $(isMax)$ **then** ▷ *a_i has the best possible improvement*
        $x_i \leftarrow bestValue$
    end if
    if $((cost > 0) \land (totalImprovement \leqslant 0)$ **then** ▷ *local minima !!*
        INCREASEWEIGHTVIOLATEDCONSTRAINTS($notRespectedConstraints$, $constraintsWeights$)
    end if
    if $((cost = 0) \land (totalOtherCosts = 0)$ **then** ▷ *all constraints are respected locally*
        CHECKIFENDOFSEARCH
    end if
    $totalOtherCosts \leftarrow 0$
    for all $a_j \in neighborgs_{a_i}$ **do**
        ASK(a_j, "receiveAffectation(a_i, x_i, $constraintsWeights[j]$)")
    end for
end procedure

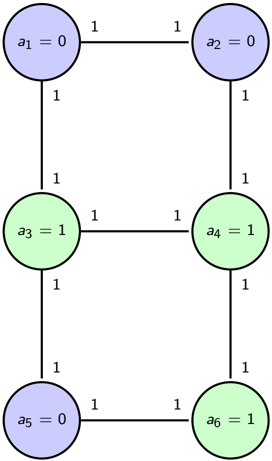
Distributed BreakOut : Exemple



Exemple de Distributed BreakOut

Soit 6 agents $a_1, a_2, \ldots a_6$, dont les variables respectives $x_1, x_2, \ldots x_6$ prennent leurs valeurs dans $\{0, 1\}$ en respectant la contraintes que deux voisins ont des valeurs différentes.

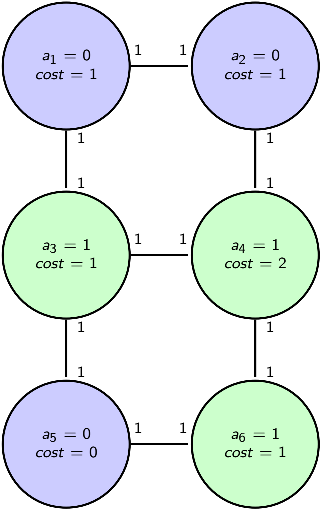
Distributed BreakOut : Exemple



Exemple de Distributed BreakOut

1. initialement, les agents choisissent une valeur aléatoirement (bleue pour 0, verte pour 1)

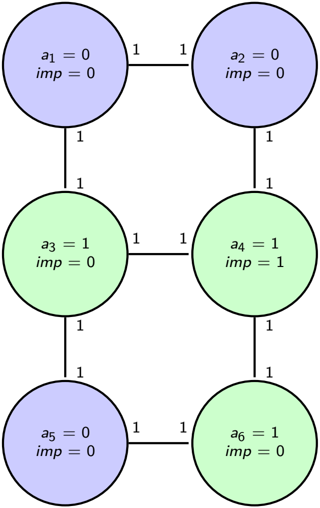
Distributed BreakOut : Exemple



Exemple de Distributed BreakOut

2. réception des affectations des voisins, identification des contraintes non respectées, calcul du coût

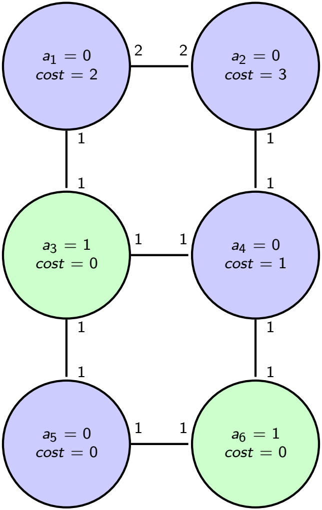
Distributed BreakOut : Exemple



Exemple de Distributed BreakOut

3. calcul de l'amélioration maximale possible, et envoi de la valeur d'amélioration
4. au vu des contraintes, a_4 gagne à changer, a_3 perdrait à changer
5. a_4 est donc l'agent effectuant le changement de valeur pour diminuer son coût de violation de contraintes..
6. a_1 est coincé dans un minimum local ($cost > 0$ et $totalImprovement \leqslant 0$), il augmente le poids de ses contraintes non respectées

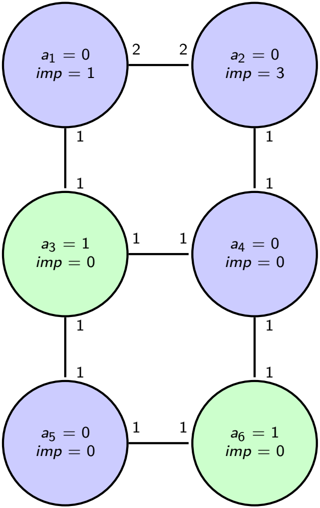
Distributed BreakOut : Exemple



Exemple de Distributed BreakOut

7. reCalcul des coûts suite à la réception des nouvelles valeurs

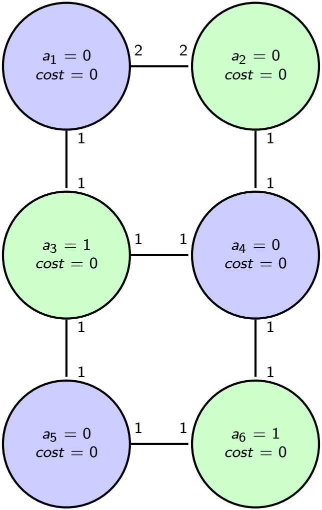
Distributed BreakOut : Exemple



Exemple de Distributed BreakOut

8. recalcul des améliorations maximales possible par chacun.
9. a_2 a la plus grande amélioration possible, il change sa valeur….

Distributed BreakOut : Exemple



Exemple de Distributed BreakOut

9. recalcul des coûts
10. La somme des coûts est nulle, une solution est trouvée. . .

Constraint Optimization Problem : DCOP

Définition d'un Problème d'Optimisation de Contraintes

Similaire au problème de type CSP, avec valeurs réelles et recherche de **la solution qui minimise les violations de contraintes**, au contraire du CSP qui tente de trouver la solution qui *respecte les contraintes*.

- Soit X un ensemble de variables x_i,
- Soit D le domaine de valeurs des variables de X,
- soit C un ensemble de contraintes c_i sur un ensemble de variables tq $c_i(x_{i0}, \ldots, x_{ij}) \rightarrow \Re$
- résoudre un COP consiste à affecter des valeurs aux variables de X afin de minimiser la somme des c_i de C

Problème NP-complet

Résolution d'un COP : Branch and Bound

Similaire à l'algorithme $A*$.

- Séparation (pouvant être récursive) d'un problème en sous-problèmes.
- Evaluation des noeuds de l'arbre de recherche.

Algorithme simple de Branch and Bound pour COP

```
c* ← +∞                                           ▷ cost of contraints violation.
g* ← ∅                                             ▷ solution : affectations
BranchAndBound(i, g)

procedure BranchAndBound(i, g)
    if  i = n then                                 ▷ last variable.
        if C(g) < c∗ then             ▷ Cost of violations in g is interesting.
            g∗ ← g
            c∗ ← C(g)
            return
        end if
    end if
    for all  v ∈ D_i  do
        g′ ← g ∪ {x_i ← v}
        if  C(g′) < c∗  then
            BranchAndBound(i + 1, g′)
        end if
    end for
end procedure
```

Distributed Constraint Optimization Problem : DCOP

Distributed Constraint Optimisation Problem : DCOP

- La distribution d'un COP consiste à définir un agent a_i par variable x_i.
- Chaque agent a_i a la responsabilité de l'affectation d'une valeur à sa variable x_i, en fonction des valeurs définies par les agents de son voisinages, obtenues par communication

Plusieurs algorithmes :

- ADOPT : <u>A</u>synchronous <u>D</u>istributed constraint <u>opt</u>imization.
- OptAPO : Optimal Asynchronous Partial Overlay.
- ...

ADOPT : Asynchronous Distributed constraint optimization

Eléments de définition : contexte et coûts

- **contexte :** une solution partielle, un ensemble d'affectations : $\{(x_j, d_j), (x_k, d_k), \ldots\}$.
- **coût local :** $\delta(d_i)$ est le coût local lorsque a_i prend la valeur d_i :
$$\delta(x_i) = \sum_{(x_j,d_j)\ \in currentContext} f_{ij}(d_i, d_j)$$
- **seuil bas fils si d :** $lb(d, x_l)$ est le seuil bas remonté par le fils x_l si x_i choisi d
- **seuil bas si d :** $LB(d) = \delta(d) + \sum_{x_l \in children} lb(d, x_l)$
- **seuil bas** sous x_i : $LB = min_{d \in D_i} LB(d)$
- **seuil haut fils si d :** $ub(d, x_l)$ est le seuil haut remonté par le fils x_l si x_i choisi d
- **seuil haut si d :** $UB(d) = \delta(d) + \sum_{x_l \in children} ub(d, x_l)$
- **seuil haut** sous x_i : $UB = min_{d \in D_i} UB(d)$

ADOPT : Asynchronous Distributed constraint optimization

Remarques sur les coûts

- $LB = k$ signifie qu'il n'est pas possible que la sommes des coûts des fils de a_i soit $< k$ étant donné les choix des parents
- $UB = k$ signifie qu'il n'est pas possible que la sommes des coûts des fils de a_i soit $> k$ étant donné les choix des parents
- pour un agent feuille : $\delta(d) = LB(d) = UB(d)$
- si x_i n'a pas reçu de coûts de ses fils, $UB = +\infty$ et $LB = \min_{d \in D_i} \delta(d)$

ADOPT : Asynchronous Distributed constraint optimization

Eléments de définition : seuil de backtracking

- **seuil de backtracking : threshold**
  - variable initialisée à 0
  - augmenté par x_i si le coût de la solution optimale dans ses fils doit être $> threshold$
  - réduite par x_i si le coût de la solution optimale dans ses fils doit être $< threshold$
  - $\rightarrow LB \leqslant threshold \leqslant UB$

Remarques sur le threshold

- a_i peut diviser son threshold et définir ainsi ceux de ses fils
- $t(d, x_l)$ est le threshold alloué par le parent x_i ayant choisi la valeur d au fils x_l.
- **AllocationInvariant :** $threshold = \delta(d) + \sum_{x_l \in children} t(d, x_l)$
- **ChildThresholdInvariant :** $\forall d \in D_i, \forall x_l \in children, lb(d, x_l) \leqslant t(d, x_l) \leqslant ub(t, x_l)$
- si $LB > threshold$, l'agent choisit une valeur moins coûteuse

ADOPT : Asynchronous Distributed constraint optimization

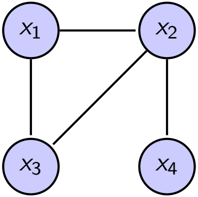
Eléments de l'algorithme ADOPT

- Les Messages
  - Messages 'Valeur' : propagés d'un agent vers ses descendants partageant une contrainte
  - Messages 'Coûts' (de violations de contraintes) : propagés vers les parents dans l'arbre
    - un agent envoie son coût cumulé, ainsi que son contexte
  - Messages 'Thresholds' (seuils) : propagés d'un agent vers ses fils dans l'arbre

ADOPT : Un exemple simple (issu de [Pragnesh, Wei-Min, Milind, Makoto, AIJ'03])

Description de l'exemple simple pour ADOPT

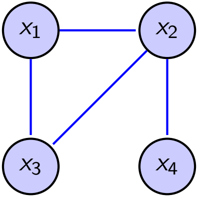
- Soient $X = \{x_1, x_2, x_3, x_4\}$, $D = \{0, 1\}$
  x_1, x_2, x_3 voisines (partageant des contraintes) ; x_2 voisine avec x_4
- Soit C l'ensemble de contraintes entre les deux valeurs de D
  $c_1(0, 0) = 1, c_2(1, 0) = 2, c_3(0, 1) = 2, c_4(1, 1) = 0$
- Donc, une résultat serait que toutes les variables prennent la valeur 1 :
  $\rightarrow \sum_i c_i = 0$

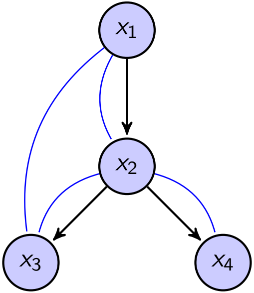


ADOPT : Un exemple simple (issu de [Pragnesh, Wei-Min, Milind, Makoto, AIJ'03])

Description de l'exemple simple pour ADOPT

1. Transformation en forme arborescente
   (plusieurs formes possibles)

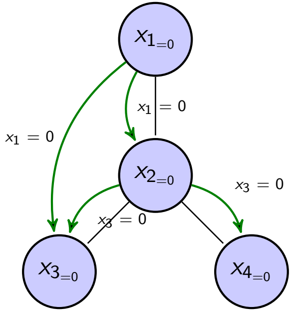


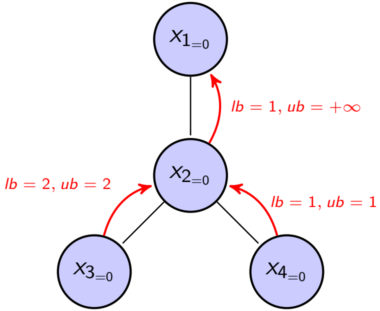


ADOPT : Un exemple simple (issu de [Pragnesh, Wei-Min, Milind, Makoto, AIJ'03])

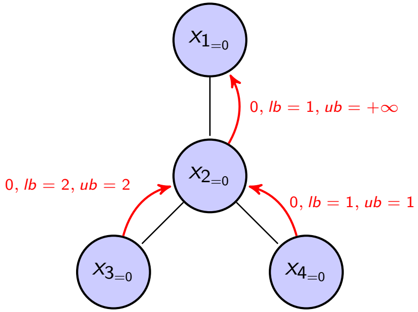
Description de l'exemple simple pour ADOPT

2. Transmission des valeurs aux fils voisins
3. Calcul réparti des coûts bas (*lb*) et hauts (*ub*) en fonction du contexte





ADOPT : Un exemple simple (issu de [Pragnesh, Wei-Min, Milind, Makoto, AIJ'03])



Calcul des coûts

x_2 : $LB(0) = \delta(0) + lb(0, x_3) + lb(0, x_4) = 1 + 0 + 0 = 1$
x_2 : $LB(1) = \delta(1) + lb(1, x_3) + lb(1, x_4) = 2 + 0 + 0 = 2$
x_2 : $UB(0) = \delta(0) + ub(0, x_3) + ub(0, x_4) = 1 + \infty + \infty = \infty$
x_2 : $UB(1) = \delta(1) + ub(1, x_3) + ub(1, x_4) = 2 + \infty + \infty = \infty$
x_2 : $LB = 1; UB = \infty$

x_3 : $LB(0) = \delta(0) = c(0, 0) + c(0, 0) = 1 + 1 = 2.$
x_3 : $LB(1) = \delta(1) = c(1, 0) + c(1, 0) = 2 + 2 = 4.$
x_3 : $LB = UB = 2.$

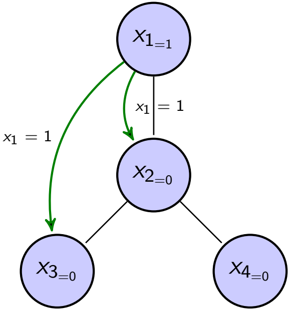
x_4 : $LB(0) = \delta(0) = c(0, 0) = 1.$
x_4 : $LB(1) = \delta(1) = c(1, 0) = 2.$
x_4 : $LB = UB = 1.$

ADOPT : Un exemple simple (issu de [Pragnesh, Wei-Min, Milind, Makoto, AIJ'03])

Calcul du coût pour x_1 et choix

x_1 : $LB(1) = \delta(1) + LB(1, x_2) = 2 + 0 = 2$
x_1 : $LB(0) = \delta(0) + LB(0, x_2) = 1 + 1 = 1$
x_1 : $UB(1) = \delta(1) + UB(1, x_2) = 2 + 0 = 2$
x_1 : $UB(0) = \delta(0) + UB(0, x_2) = 1 + \infty = \infty$
Remarque $LB(1, x_2) = 0$ *car* x_1 *n'a pas d'info dessus et suppose donc cette valeur.* a_1 choisit la valeur $x_1 = 1$

4. a_1 transmet sa valeur à ses fils

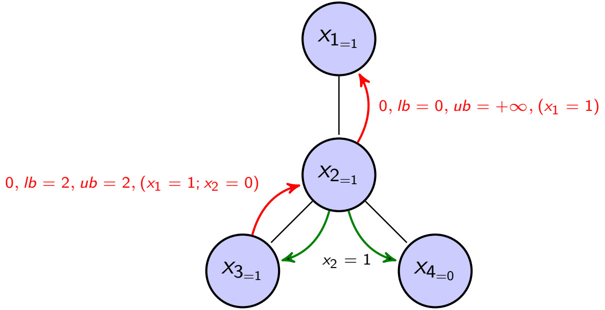


ADOPT : Un exemple simple (issu de [Pragnesh, Wei-Min, Milind, Makoto, AIJ'03])

Calcul du coût pour x_2 et x_3 et choix

5. Recalcul des coûts par x_2 et x_3

x_2 : $LB(1) = \delta(1) + LB(1, x_3) + LB(1, x_4) = 0 + 0 + 0 = 0$
x_2 : $LB(0) = \delta(0) + LB(0, x_3) + LB(0, x_4) = 2 + 2 + 1 = 5$
a_2 choisit la valeur $x_2 = 1$ et transmet sa valeur
x_3 : $LB(0) = \delta(0) = c(0, 1) + c(0, 0) = 2 + 1 = 3.$
x_3 : $LB(1) = \delta(1) = c(1, 1) + c(1, 0) = 0 + 2 = 2.$
x_3 : $LB = UB = 2. \rightarrow a_3$ choisira la valeur $x_3 = 1$

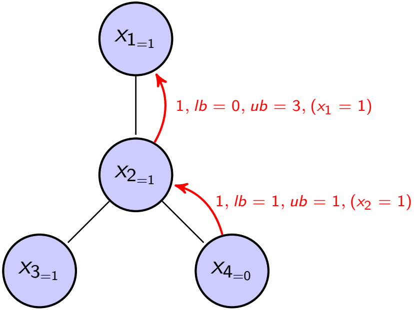


ADOPT : Un exemple simple (issu de [Pragnesh, Wei-Min, Milind, Makoto, AIJ'03])

Calcul du coût pour x_2 et x_4 et choix

6. Recalcul des coûts par x_2, x_3 et x_4

x_4 : $LB(0) = \delta(0) = c(0,1) = 2 = 2.$
x_4 : $LB(1) = \delta(1) = c(1,1) = 0 = 0.$
x_4 : $LB = UB = 0. \rightarrow a_4$ choisira la valeur $x_4 = 1$
x_2 : $UB(1) = \delta(1) + UB(1, x_3) + UB(1, x_4) = 0 + 2 + 1 = 3$
x_2 : $UB(0) = \delta(0) + UB(0, x_3) + UB(0, x_4) = 2 + 2 + 2 = 6$
a_2 choisit la valeur $x_2 = 1$ avec $LB = 0$ et $UB = 3$

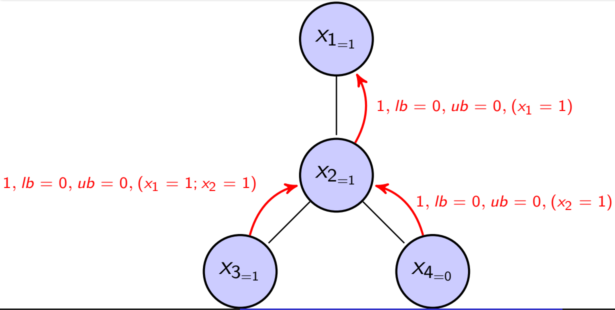


ADOPT : Un exemple simple (issu de [Pragnesh, Wei-Min, Milind, Makoto, AIJ'03])

Calcul du coût pour x_2, x_3 et x_4

7. Recalcul des coûts par x_2, x_3 et x_4

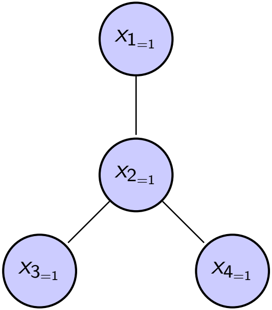
x_3 : $LB(0) = \delta(0) = c(0,1) \times 2 = 4$. $LB(1) = \delta(1) = c(1,1) \times 2 = 0$.
x_3 : $LB = UB = 0$. $\rightarrow a_3$ transmet ses coûts.
x_4 : $LB(0) = \delta(0) = c(0,1) = 2$. $LB(1) = \delta(1) = c(1,1) = 0$.
x_4 : $LB = UB = 0$. $\rightarrow a_4$ transmet ses coûts
x_2 : $UB(1) = \delta(1) + UB(1, x_3) + UB(1, x_4) = 0 + 0 + 0 = 0$
x_2 : $UB(0) = \delta(0) + UB(0, x_3) + UB(0, x_4) = 2 + 4 + 2 = 8$
a_2 choisit la valeur $x_2 = 1$ avec $LB = 0$ et $UB = 0$



ADOPT : Un exemple simple (issu de [Pragnesh, Wei-Min, Milind, Makoto, AIJ'03])

Description de l'exemple simple pour ADOPT

- Tous les coûts sont nuls
  $\rightarrow$ Aucune amélioration possible $\rightarrow$ Fin de l'algorithme



Exemple d'algorithmes pour ADOPT

adaptés de 'Fundamentals of Multiagent Systems with NetLogo Examples', José M Vidal,March 1, 2010
Pour un agent a_i donné :

procedure $\textsc{ResetVariables}(d, child)$
  $lowerBound\,[d, child] \leftarrow 0$
  $t\,[d, child] \leftarrow 0$
  $upperBound\,[d, child] \leftarrow +\infty$
  $context\,[d, child] \leftarrow \{\}$
end procedure

procedure $\textsc{Initialise}$
  $threshold \leftarrow 0$
  $receivedTerminate \leftarrow False$
  $currentContext \leftarrow \{\}$
  for all $d \in D_i$ **do**
    for all $child \in children_i$ **do**
      $\textsc{ResetVariables}(d, child)$
    end for
  end for
  $x_i \leftarrow d \in D$, such as x_i minimizes $\left(cost_i + \sum_{child \in children} lowerBound\,[d, child]\right)$
  $\textsc{BackTrack}$
end procedure

Exemple d'algorithmes pour ADOPT

procedure HANDLETHRESHOLD(t, $context$)
    if ISCOMPATIBLEWITH($context$, $currentContext$) **then**
        $threshold \leftarrow t$
        MAINTAINTHRESHOLDINVARIANT
        BACKTRACK
    end if
end procedure

procedure HANDLETERMINATE($context$)
    $receivedTerminate \leftarrow True$
    $currentContext \leftarrow context$
    BACKTRACK
end procedure

procedure MAINTAINTHRESHOLDINVARIANT

$$b = min_{d \in D_i}(cost(d) + \sum_{child \in children} lowerBound\,[d, child])$$

    if $threshold < b$ **then**
        $threshold \leftarrow b$
    end if

$$u = min_{d \in D_i}(cost(d) + \sum_{child \in children} upperBound\,[d, child])$$

    if $threshold > u$ **then**

Exemple d'algorithmes pour ADOPT

procedure HANDLEVALUE(j, x_j)
  if $\neg receivedTerminate$ **then**
    $currentContext\,[j] \leftarrow x_j$
    for all $d \in D_i, child \in children_i$ **do**
      if $\neg$ISCOMPATIBLEWITH($context\,[d, child]\,, currentContext$) **then**
        RESETVARIABLES(d, child)
      end if
    end for
    MAINTAINTHRESHOLDINVARIANT
    BACKTRACK
  end if
end procedure

procedure MAINTAINALLOCATIONINVARIANT
  while $threshold > (cost(x_i) + \sum_{child \in children} t\,[x_i, child])$ **do**
    $chosen \leftarrow child' \in children_i\ suchAs\ upperBound\,[x_i, child'] > t\,[x_i, child']$
    $t\,[xi, chosen] \leftarrow t\,[xi, chosen] + 1$
  end while
  while $threshold < (cost(x_i) + \sum_{child \in children} t\,[x_i, child])$ **do**
    $chosen \leftarrow child' \in children_i\ suchAs\ lowerBound\,[x_i, child'] < t\,[x_i, child']$
    $t\,[xi, chosen] \leftarrow t\,[xi, chosen] - 1$
  end while
  $\forall child \in children_i, child$.HANDLETHRESHOLD($t\,[x_i, chosen]\,, currentContext$)

Exemple d'algorithmes pour ADOPT

```
procedure MAINTAINCHILDTHRESHOLDINVARIANT
    for all d ∈ D_i, child ∈ children_i do
        if lowerBound[d, child] > t[d, child] then
            t[d, child] ← lowerBound[d, child]
        end if
    end for
    for all d ∈ D_i, child ∈ children_i do
        if upperBound[d, child] < t[d, child] then
            t[d, child] ← upperBound[d, child]
        end if
    end for
end procedure
```

Exemple d'algorithmes pour ADOPT

procedure BACKTRACK
  if $threshold = min_{d \in D_i}(cost(d) + \sum_{child \in children} upperBound\,[d, child])$ **then**
    $x_i \leftarrow argmin_{d \in D}(cost(d) + \sum_{child \in children} upperBound\,[x_i, child])$
  else if $threshold < min_{d \in D_i}(cost(d) + \sum_{child \in children} lowerBound\,[d, child])$ **then**
    $x_i \leftarrow argmin_{d \in D}(cost(d) + \sum_{child \in children} lowerBound\,[x_i, child])$
  end if
  $(\forall k \in neighbors_i | k.priority < piority)$ k.HANDLEVALUE(i, x_i)
  MAINTAINALLOCATIONINVARIANT
  if $threshold = min_{d \in D_i}(cost(d) + \sum_{child \in children} upperBound\,[d, child])$ and $(receivedTerminate$ or $isRoot)$ **then**
    $currentContext\,[i] \leftarrow x_i$
    $\forall child \in children_i, child$.HANDLETERMINATE$(currentContext)$
    EXIT
  end if
  $parent$.HANDLECOST$(i,\ currentContext,$ $min_{d \in D_i}(cost(d) + \sum_{child \in children} lowerBound\,[d, child]),$ $min_{d \in D_i}(cost(d) + \sum_{child \in children} upperBound\,[d, child]))$
end procedure

Exemple d'algorithmes pour ADOPT

procedure HANDLECOST(k, $context$, lb, ub)
  $d \leftarrow context[i]$
  DELETE($context[i]$)
  if $\neg receivedTerminate$ **then**
    for $(j, x_j) \in context | j \notin neighbors_j$ **do**
      $currentContext[j] \leftarrow x_j$
    end for
    for all $d' \in D_i, child \in children_i$ **do**
      if $\neg$ISCOMPATIBLEWITH($context[d', child]$, $currentContext$) **then**
        RESETVARIABLES(d', child)
      end if
    end for
    if ISCOMPATIBLEWITH($context$, $currentContext$) **then**
      $lowerBound[d, k] \leftarrow lb$
      $upperBound[d, k] \leftarrow ub$
      $context[d', k] \leftarrow context$
      MAINTAINCHILDTHRESHOLDINVARIANT
      MAINTAINTHRESHOLDINVARIANT
    end if
    BACKTRACK
  end if
end procedure