# Valued Dynamic Backtracking Distribué

Mustapha Belaïssaoui

ENCG, Université Hassan 1er
Km 3, Route de Casa, B.P 658, Settat
Maroc
Email : m.belaissaoui@encg-settat.ma

**RESUME.** Nous proposons une nouvelle méthode pour résoudre des Problèmes de Satisfaction de Contraintes Valués et Distribués (DisVCSP). Cette méthode utilise à la fois certaines des bonnes propriétés de la version centralisée de Dynamic Backtracking Valué (VDB)[1] et la méthode Dynamic Backtracking Distribué (DisDB)[2]. Elle vise à bénéficier des avantages des deux approches : la gestion des nogoods valués, la complétude de la recherche et un haut niveau d'asynchronisme entre les agents.

**ABSTRACT.** We propose a framework for solving Distributed Valued Constraint Satisfaction Problems (DisVCSP). This approach permits us to define a new framework for the enumeration, which we expect that it will benefit from the advantages of two approaches : recording and exploiting the valued nogoods in Valued Dynamic Backtracking (VDB)[1] ; the completness of search and the asynchronisme between agents in Distibuted Dynamic Backtracking (DisDB)[2].

**MOTS-CLES.** Problèmes de Satisfaction de Contraintes Valués et Distribués, IA Distribué.

**KEY WORDS.** Distributed Valued Constraint Satisfaction Problems, Distributed AI.

## 1. Introduction

Dans le cadre des CSP, la recherche d'une solution requiert de satisfaire toutes les contraintes. Dans la mesure où certaines contraintes doivent être obligatoirement satisfaites, elles sont *"dures"*. Cependant, pour certains problèmes réels, certaines contraintes *"molles"* ne traduisent, dans la réalité, qu'une préférence, une possibilité... Afin de pouvoir exprimer de telles contraintes, plusieurs extensions des CSP ont été proposées parmi lesquelles les CSP valués (***VCSP***) [3][4][5]. Une ***valuation*** est associée à chaque contrainte. La valuation d'une contrainte traduit l'importance de la violation de cette contrainte. La méthode de base pour la résolution de VCSP est l'algorithme Branch and Bound. De nombreuses améliorations ont été proposées, notamment à partir du cadre CSP et ont conduit à des méthodes comme les algorithmes Forward-Checking valué [3], Nogood Recording [13], la recherche arborescente bornée pour la résolution de CSP valué [5], la méthode RDS [6], les méthodes issues de la programmation dynamique [7]... Les ***DisCSP*** consistent à représenter les systèmes à contraintes en agents [8]. Les algorithmes complets pour la résolution de DisCSP doivent beaucoup à Yokoo et à ses collaborateurs, pionniers du domaine avec ABT et AWC [9][10]. Plus récemment, ODIBT est présenté dans [12]. C'est une version complète et optimale en envois de messages de DIBT [11].

Dans cet article, nous proposons une nouvelle méthode pour le modèle DisVCSP. Notre but en l'écrivant était de garder le meilleur de *DisDB* [2], pour obtenir un algorithme aussi asynchrone que possible, qui minimise l'envoi de messages entre agents indépendants tout en restant complet. Notre algorithme a aussi une parenté évidente avec *VDB* [1]. Nous avons donc gardé les nogoods de ce dernier.

## 2. Définitions préliminaires

Un CSP est un (X, D, C). $X=\{x_1, .... x_n\}$, variables. $x_i$ prent ses valeurs dans un domaine fini $D_{xi} \in D$. Ces variables sont soumises à des contraintes issues de C. Chaque contrainte $c=\{x_{c1},.., x_{ck}\}$ est l'ensemble des tuples autorisés sur $D_{xc1}x..xD_{xck}$. Une ***affectation partielle*** est un sous-ensemble de variables affectées. Lorsque toutes les variables sont affectées, l'affectation est dite ***complète***. Deux affectations sont ***compatibles*** si les variables communes sont affectées aux mêmes valeurs. Une ***structure de valuation*** [3] est un triplet $(E, \preceq, \oplus)$ avec un ensemble E de valuations totalement ordonné par $\preceq$, muni d'un élément minimum $\bot$, d'un élément maximum $\top$ et d'une loi de composition interne $\oplus$ commutative, associative, monotone. $\bot$ est un élément neutre pour $\oplus$ et T est un élément absorbant. Un ***VCSP*** [3] est un CSP doté d'un $S=(E,\preceq,\oplus)$ et d'une application $\phi$ de C dans E, qui associe une valuation à chaque contrainte. On le note $(X, D, C, S, \phi)$.

**Définition 1** ([3]) Soient un VCSP P = (X, D, C, S, $\phi$) et une instanciation B sur X. La ***valuation*** de B dans P se définit par : $v_P(B)=_{c \in C} \oplus_{B \text{ viole } c} \phi(c)$. P consiste donc à trouver une affectation complète qui soit de ***valuation minimale*** $\alpha$.

**Exemple** Malek, Badr et Ali se rencontrent dans un couloir. Malek interpelle les deux autres et leur rappelle leur promesse de présentation de leurs travaux respectifs. Badr et Ali doivent présenter leurs travaux à Malek qui lui même doit leur présenter ses travaux. Il y a 4 exposés : de Badr à Malek, d'Ali à Malek, de Malek à Badr et enfin de Malek à Ali. Chaque exposé prend une demi-journée. Malek signale qu'il lui paraît important d'avoir écouté Badr et Ali avant de faire ses présentations. Puis, Malek indique qu'il aimerait présenter ses travaux à Badr lors de la 2ème demi-journée. Enfin, Malek signale qu'il ne veut pas présenter ses travaux à Badr et Ali en même temps.

**Modélisation** $X=\{M_b, M_a, B_m, A_m\}$, où $M_b$ désigne l'exposé que doit faire Malek à Badr… Les domaines sont les 4 demi-jounées : $D_x=\{1,2,3,4\}$. ***Contraintes d'intégrité*** : un orateur ne peut être auditeur et on ne peut assister à deux présentations en même temps : $C_1$:$M_a \neq A_m$, $C_2$:$M_b \neq B_m$, $C_3$:$M_a \neq B_m$, $C_4$:$M_b \neq A_m$ et $C_5$:$A_m \neq B_m$. ***Contraintes de préférence*** : Malek signale qu'il lui paraît important d'avoir écouté Badr et Ali avant de faire ses présentations: $C_6$:$M_a > A_m$, $C_7$:$M_a > B_m$, $C_8$:$M_b > B_m$ et $C_9$:$M_b > A_m$. Malek aimerait présenter ses travaux à Badr lors de la 2ème demi-journée et il ne veut pas présenter à Badr et Ali en même temps : $C_{10}$ :$M_b=2$ et $C_{11}$:$M_a \neq M_b$. $S=(\mathbb{N}, <, +)$ avec $\perp=0$ et $T=+\infty$. 3 niveaux de violations : pour chaque contrainte d'intégrité, la valuation est de 3. Les contraintes d'antériorité ont un poids de 2 et les contraintes de simultanéité ($C_{10}$ et $C_{11}$) ont une valuation de 1. $\alpha=1$ (il suffit de violer $C_{10}$ pour avoir une solution).

**Définition 2** ([3]) Soient un VCSP P et une instanciation B sur $Y \subset X$. La ***valuation locale*** de B dans P se définit par :$v(B)= v(B)=_{c \in C / c \subset \zeta} \oplus_{B \text{ viole } c} \phi(c)$. Nous notons $v_\zeta(B)$ la valuation de B sur le problème restreint au sous-ensemble de contraintes $\zeta \subset \mathbf{C}$.

**Définition 3** Un ***DisVCSP*** est un VCSP où les variables, valeurs, contraintes et valuations sont distribuées parmi un ensembe d'agents $A_i$. Chaque $A_i$ a un VCSP $(X_i,D_i,C_i,S,\phi)$. Un ***DisVCSP*** est un $P=(X, D, C, S, \phi, A)$ où $A=\{A_1,.., A_p\}$. Les DisVCSP sont résolus par l'action collective et coordonnée des agents de A [10], chacun exécutant un processus de satisfaction de contraintes. Pour des raisons de simplicité, dans la suite de cet article nous supposons que chaque agent a une seule variable.

## 3. Valued Dynamic Backtracking Distribué (DisVDB)

VDB [1] mémorise les zones de l'espace de recherche explorées et reconnues comme sans solution sous la forme des ***nogoods***. Afin de s'adapter à ce contexte, nous rappelons ce concept ainsi que les propriétés [13] qui permettent sa mise en oeuvre.

**Définition 4** Un ***Nogood valué*** est un triplet ***(B, v, C)***. *B* est une affectation partielle, *v* une valuation et *C* un ensemble de contraintes (***justification***) tel que, pour toute affectation *B'* extension complète de *B*, $v \leq v_C(B')$.

**Propriété 1** ***([13] construction)*** Soit *B* une affectation partielle, et *C* un ensemble de contraintes qu'elle viole, alors *(B, $v_C(B)$, C)* est un nogood.

**Propriété 2** ***( [13] union)*** Soit *B* une affectation partielle, et $x^1, .., x^m$ les valeurs d'une variable $x \notin B$. Soient $B_1,...,B_m$ des sous-ensembles de *B* tels que $(B_1 \cup \{x^1\}, v_1, C_1),.., (B_m \cup \{x^m\}, v_m, C_m)$ sont des nogoods. Alors $(\cup B_i, \min(v_i), \cup C_i)_{i \in \{1,..,m\}}$ est un nogood.

**Propriété 3** ***( [13] projection)*** Soit *(B, v, C)* un nogood, et *B'* l'ensemble des valeurs de *B* absentes des n-uplets interdits par les contraintes de C. *(B-B', v, C)* est un nogood.

**Propriété 4** ***([13] augmentation)*** Soit *(B, v, C)* un nogood, *B'* une affectation compatible avec *B* et *c* violée par *B'*. Si $c \notin C$, $(B \cup B', v \oplus \phi(c), C \cup \{c\})$ est un nogood.

**Propriété 5** ***([13] réduction)*** Soit *(B, v, C)* un nogood, et *c* une contrainte de *C*. Soit $v' = min(\varepsilon \in E / \varepsilon \oplus \phi(c) \geq v)$, alors *(B, v', C-{c})* est un nogood. Soit *B* une affectation et *n* un nogood d'affectation compatible, nous noterons $\uparrow^{nB}(n)$ le résultat des augmentations successives de *n* avec toutes les contraintes violées par *B*, et $\downarrow^{lB}(n)$ le résultat des réductions successives du nogood *n* avec toutes les contraintes violées par *B*.

**Propriété 6** ***( [1] équivalence)*** Soit $n=(B, v, C)$ et $c \in C$ violée par une *B'* compatible. Le nogood *n'* obtenu par réduction puis augmentation de *n* avec *c* possède une valuation supérieure ou égale à celle de *n*. Donc, la valuation d'un nogood *n* est inférieure ou égale à celle de $\uparrow^{nB}(\downarrow^{lB}(n))$.

**Définition 5** [1] Soit *n* un nogood portant sur *B*. Soit $C_{B \cap n}$ : contraintes violées par *B* et présentes dans *n*, et $C_{B-n}$ : contraintes violées par *B*. Nous appelons ***réduction partielle*** $\Downarrow_{(B, n)}$ d'un nogood par *B* et *n* l'augmentation avec les contraintes de $C_{B \cap n}$ suivie de la réduction avec les contraintes de $C_{B-n}$ . Il a pour effet de projeter un nogood *n'* de manière à ce que le résultat *n''* ait une valuation supérieure à son prédécesseur *n*.

***VDB*** [1] mémorise pour chaque $i^{ème}$ valeur $xi \in D_x$ un nogood $N_{xi}$ donnant un minorant de la valuation globale de l'extension d'une affectation *B* avec cette valeur. VDB choisit une variable non instanciée, et tente de lui affecter une valeur. Si cette valeur et les variables déjà instanciées, donnent une valuation supérieure à $\alpha$, elle est supprimée et le nogood correspondant est mémorisé. Lorsque toutes les valeurs de la variable courante *x* sont éliminées, les justifications de retrait sont utilisées pour générer un nouveau nogood comme suit. Soit *B* l'ensemble des variables déjà instanciées ($x \notin B$). Le nouveau ***n*** est l'union des nogoods de valuations maximales de $B \cup \{xi\}$, $\forall xi \in D_x$. Pour avoir un nogood de valuation maximale, il suffit d'utiliser la propriété

d'augmentations successives de $N_{yj}$ avec toutes les contraintes violées par $B \cup \{xi\}$, $\forall xi \in D_x$ et $\forall yj \in B \cup \{xi\}$ : $\boldsymbol{n=Union(\Gamma^{B\cup\{xi\}}(N_{yj}),\ \forall xi \in D_x, \forall yj \in B \cup \{xi\})}$. VDB impose des conditions d'ordre entre les variables du nogood et la variable sur laquelle se porte le backtrack. Soit $yj \in B$ une valeur qui ne soit inférieure à aucune autre valeur de $\boldsymbol{n}$ selon l'ordre partiel. Les $N_{yk}$, dont l'affectation contient $yj$, sont initialisés par $\boldsymbol{n_{\varnothing}=(\varnothing,\perp,\varnothing)}$. $N_{yj}$ est initialisé par $\boldsymbol{\Downarrow_{(B,N_{yj})}n}$. yj est supprimée de B et VDB cherche à réinstancier y.

***DisVDB*** (voir algorithme ci-dessous) place un ordre total entre les agents. Dans ce sens, il utilise la méthode d'ordonnancement EDisAO [12]. EDisAO prend en entrée $\Gamma$, $f$ et ***op***. $\Gamma$, ensemble des agents du système avec lesquels ***self***, ***l'agent générique***, partage au moins une contrainte. $f$, la fonction heuristique et ***op***, l'opérateur de comparaison. Il construit une hiérarchie d'agents : $\boldsymbol{\Gamma^-(self)}$, agents classés plus haut dans la hiérarchie. $\boldsymbol{\Gamma^+(self)}$, agents fils. Chacun des agents exécute ***DisVDB*** et mémorise un contexte et des nogoods. Les agents échangent des affectations et des nogoods. Les affectations sont toujours acceptées et le contexte mis à jour en conséquence. Un nogood est accepté s'il est cohérent avec le contexte de l'agent et sa propre valeur, sinon il est ***obsolète***. S'il est accepté, il est mémorisé comme justification du retrait de la valeur. Lorsque le domaine d'un agent serra vide, DisVDB procède comme dans VDB. Les messages échangés sont de trois types : ***Stop(system)***, il n'y a pas de solution. Ce message implique un agent supplémentaire appelé ***system***, qui est responsable de l'arrêt du réseau. ***Info***$(A_i, v, N_v)$, informe $A_i \in \Gamma^-$(self) que self a pris la valeur $v$ et que le nogood généré pour cette valeur est $N_v$, c'est pour que $A_i$ peut calculer la valuation de son nogood généré, s'il n'arrive pas à instancier sa variable. ***Back(nogood, $A_j$)***, Backtrack. Il contient un nogood et est adressé à $A_j$. $A_j$ est le dernier agent qui a envoyé un message de type info. Par ce que, self croit que ce dernier est la cause du retrait. Les messages sont échangés via les primitives ***getMsg*** et ***sendMsg***. ***DisVDB*** est une boucle de réception qui commute l'exécution suivant le type de message reçu. Après réception d'un message ***Stop*** de l'agent S***ystem***, elle s'achève. $\boldsymbol{\alpha}$ est fournie par la dernière solution trouvée et fixée a priori à la valuation maximale. A chaque valeur $i$ de la variable de self est associé un nogood $\boldsymbol{N_i}$. C'est une mémoire des anciennes tentatives infructueuses d'extension avec $i$. Ces $\boldsymbol{N_i}$ sont initialisés par $\boldsymbol{n_{\varnothing}=(\varnothing,\perp,\varnothing)}$. ***GoAhead*** est exécutée lors de la réception d'un message de type ***Info***. Elle tente d'étendre le contexte reçu par self. Après une mise à jour de ***myContext*** (1), si ***myValue*** (2) est supprimée par ***Evalue***, GoAhead fait appel à ***ChooseValue*** pour trouver une autre valeur respectant myContext (3). Si c'est possible, self informe $\boldsymbol{\Gamma^+(self)}$ (4). Sinon, self appelle ***ResolveNogoods*** (5). Si l'argument de GoAhead est vide, elle essaye de trouver une valeur respectant myContext (DisVDB, 2 et ResolveNogoods, 10). Si le Backtrack n'est pas obsolète, ***ResolveConflict*** actualise myContext (2). Elle construit un nouveau nogood, par la définition de ***la réduction partielle*** (3, 4). Ensuite, elle tente de trouver une autre valeur. Si c'est possible, self informe $\boldsymbol{\Gamma^+(self)}$ (6). Sinon, elle fait appel à ***ResolveNogoods*** (7). Si le Backtrack respecte seulement la valeur de self, self renvoie cette valeur à l'expéditeur (8, 9).

**Algorithm : Distributed valued Dynamic Backtracking**

```
Procedure DisVDB()
1 EDisAO(Γ, f, op) ;
2 GoAhead(null) ;
3 end ← false ;
4 while not(end) do
5         msg ← getMsg() ;
6         switch(msg.type)
7                 Stop      : end ← true ;
8                 Info      : GoAhead(msg) ;
9                 Back      : ResolveConflict(msg) ;
Procedure GoAhead(msg)
1 if (msg) then Update (myContext, msg. Context) ;
2 if (Not(msg) or Valuation(Evalue(myContext ∪{myValue})) > α) then
3         myValue ← ChooseValue() ;
4         if (myValue) then for each child ∈Γ⁺(self) do sendMsg:Info(child,myValue,N_myValue );
5         else ResolveNogoods() ;
Procedure ResolveConflict(msg)
1 if Not(obsolete (msg.Context on Γ⁻ (self) ∪ {self }, myContext)) then
2         Update (myContext,msg. Context) ;
3         N_myValue ← ⋃_{msg.context.hisValue} (n) ;
4         myNogoods ← myNogoods ∪{N_myValue};
5         myValue ← ChooseValue() ;
6         if (myValue) then for each child ∈Γ⁺(self) do sendMsg:Info(child,myValue,N_myValue );
7         else ResolveNogoods();
8 else if Not(obsolete (msg.Context on self, myValue)) then
9         SendMsg:Info(msg.sender, myValue, N_myValue);
Procedure ResolveNogoods()
1 n←Union (Evalue(myContext ∪{i})) ; /* i ∈ D_self */
2 if Affectation(n)= ∅ then
3         end ← true ;
4         sendMsg:Stop(system) ;
5 else
6         Let A' ∈Value(myContext ∩ last(msg.type=info))
7         sendMsg:Back(n, A);
8         Update(myContext, myContext\{A'});
9         Update(myContext, mycontext\ ∪{x'∈Affectation(n)/myContext(Γ⁻(self))};
10        GoAhead(null);
Function ChooseValue()
1 for each v ∈ D(self) not eliminated by myNogoods do
2         if Valuation(Evalue(myContext ∪ {v})) < α then return (v) ;
3         else N_v←Evalue(myContext∪{v});
4         myNogoods ← myNogoods ∪{N_v};
5 return (∅) ;
Procedure Update (myContext, new Context)
1 revise (myContext, newContext) ;
2 for each n ∈ myNogoods do
3         if obsolete(Affectation(n), myContext) then myNogoods ← myNogoods\ {n};
Function Evalue(Context)
Return(Nogood(maximal(valuation in { ↑ Context ( N _ ), ∀ xi ∈ Context } )))
```

***ResolveNogoods*** utilise la propriété de ***l'union*** sur les nogoods générés par la fonction ***Evalue*** appliquée sur chaque valeur de $D_{self}$ et myContext. Elle génère un nouveau ***n***. Si l'affectation de ce nogood est vide, le problème est insoluble et un message de ***Stop*** est envoyé (4). Sinon, ResolveNogoods localise l'agent ***A*** qu'a envoyé

le dernier message de type Info. Elle envoie à ***A*** un message de Backtrack contenant ***n*** (7). Self oublie l'instanciation de ***A*** ainsi que toutes les affectations de ***n*** n'appartenants pas aux agents de $\Gamma^-(self)$. Aussi les nogoods locaux qui sont obsolètes. ***ChooseValue*** choisit une valeur respectant $\alpha$ avec myContext. Si une valeur est supprimée par la fonction ***Evalue***, qui fournit sous forme de nogoods une estimation des valuations globales (propriétés 2 et 4), un nouveau nogood est généré par cette dernière. ***myNogoods*** est actualisé. Si $D_{self}$ est vide, la fonction renvoie le vide.

## 4. Validité et complétude

Nous vérifions ici que l'arrêt qui n'est pas dû à l'obtention d'une meilleure solution (***ResolveNogoods,*** 2) correspond à l'absence de solutions de valuation inférieure à $\alpha$. ***Evalue(myContext∪{i})*** renvoie au moins la valuation locale ***v(myContext∪{i})***. Le test (2) dans ***ChooseValue*** assure que toute affectation complète produite possède une valuation strictement inférieure à $\alpha$. Il faut constater que le nogood ***n*** produit dans ***ResolveNogoods*** (1), comme tous les nogoods dont il est issu, possède une valuation supérieure ou égale à $\alpha$. En cas d'arrêt, on dispose d'un nogood ***n*** d'affectation vide et de valuation supérieure ou égale à $\alpha$. Ce nogood est la preuve que toute solution possède une valuation inférieure à $\alpha$. La ***terminaison*** est la propriété qui manque à la complétude de cet algorithme. Elle réside dans le choix de la valeur sélectionnée comme cible du backtrack (***ResolveNogoods***, 5). Les nogoods produits par union ont toujours une valuation supérieure ou égale à $\alpha$. Ils correspondent donc à des nogoods classiques interdisant une affectation. Pour assurer la terminaison, il suffit d'appliquer la méthode EDisAO [12] pour établir un ordre partiel entre les agents responsables des variables contenues dans le nogood ***n***. Si le choix du point de backtrack se fait de telle manière que l'ordre partiel établi par EDisAO est respecté, alors le processus termine [12][2].

## 5. Conclusion

Dans le domaine des DisVCSP, nous avons consacré notre étude à la distribution de VDB [1] et la valuation de DisDB [2]. C'est à dire, nous avons développé une méthode permettant de résoudre les DisVCSP : Distributed Valued Dynamic Backtracking (DisVDB). Nous avons d'abord défini comme algorithme d'ordonnancement des agents EDisAO [12]. EDisAO est utilisée pour accroître les performances des algorithmes de résolution des DisCSP. DisVDB a visée à bénéficier des avantages des deux approches : la gestion des nogoods valués, la complétude de la recherche et un haut niveau d'asynchronisme entre les agents. Il a été prouvé que cet algorithme est correct et complet.

## 6. Bibliographie


[1] P. Dago. Backtrack Dynamique valué, 6ème journées francophones de programmation par contraintes (JFPLC), 26-28 Mai, 1997, Orléans, France.

[2] C. Bessière, A. Maestre et P. Meseguer, Distributed dynamic backtracking. In M. Silaghi, editor, Proceedings of the IJCAI'01 workshop on distributed constraint reasoning, Seattle, pages 9-16, 2001.

[3] T. Schiex, H. Fargier, et G. Verfaillie. Valued Constraint Satisfaction Problems : hard and easy problems. Dans *Proceedings of IJCAI-95*, pages 631-637, 1995.

[4] T. Schiex, H. Fargier, et G. Verfaillie. Problèmes de Satisfaction de Contraintes Valués, Revue d'IA, Volume 11, Numéro 3, 1997.

[5] P. Jégou et C. Terrioux. Recherche arborescente bornée pour la résolution de VCSP, JNPC, 2003, Amiens, France.

[6] G. Verfaillie, M. Lemaître, et T. Schiex. Russian Doll Search for Solving Constraint Optimization Problems. Dans Proceedings of AAAI-96, pages 181-187,1996.

[7] A. Koster. Frequency Assignment - Models and Algorithms. PhD thesis, University of Maastricht, Novembre 1999.

[8] Yokoo, M., Ishida, T., et Kubawara, K. (1990). Distributed constraint satisfaction for DAI problems. In Huhns, M. N., editor, Proc. of *the 10th International Workshop on Distributed Artificial Intelligence*, chp 9.

[9] Yokoo, M., Durfee, E., Ishida, T., et Kawabara, K. (1992). Distributed constraint satisfaction for formalizing distributed problem solving. In *12th Int. Conf. on Distributed Computing Systems*, 614-624.

[10] Yokoo, M. et Hirayama, K. (1998). Distributed constraint satisfaction algorithm for complex local problems. In *ICMAS*, 372-379.

[11] Y. Hamadi, Traitement des problèmes de satisfaction de contraintes distribués, thèse de doctorat, université Montpellier II, 99.

[12] Belaissaoui, M, Contribution à l'étude de Raisonnement Temporel et au Traitement des Problèmes de Satisfaction de Contraintes Distribués, thèse de doctorat, université Mohammed V Agdal, Faculté des Sciences Rabat, 2002.

[13] P. Dago et G. Verfaillie. Nogood Recording for Valued Constraint Satisfaction Problems. Proceedings of ICTAI'96, pages 132-139, 1996.