Heuristique de choix de variable dirigée par les conflits pour la méthode hybride FC-CBJ et mémorisation de conflits pour un CSP Dynamique de T CSPs

F. Bessaïh **B.Cabon** **Ph.Michelon**

EADS ASTRIUM
, 31 rue des Cosmonautes, 31402 Toulouse, France.
Laboratoire Informatique d'Avignon
, (EA 931), F-84911 Avignon, France.
{fawzi.bessaih,philippe.michelon}@univ-avignon.fr,bertrand.cabon@astrium.eads.net

Résumé

Dans cet article, nous proposons une nouvelle heuristique de choix de variable dynamique, $dom/wdeg_{cbj}$, pour une méthode hybride alliant un schéma prospectif et un schéma rétrospectif, mis en œuvre respectivement par le *Forward-Checking* (FC) et le *Conflict Directed Backjumping* (CBJ). C'est une heuristique dirigée par les conflits et inspirée de l'heuristique *dom/wdeg* proposée par Boussemart[11] pour des méthodes prospectives. Cette nouvelle heuristique permet de réduire le phénomène appelé thrashing en choisissant en priorité les variables conduisant le plus souvent à des situations de conflit. Une telle information est enregistrée en associant un poids à chaque variable. Lors d'une situation de conflit nécessitant un retour arrière, ce poids est incrémenté pour la variable courante et la variable responsable du conflit (identifiée par CBJ). Nous proposons également, dans le cadre d'un CSP dynamique de T CSPs, un mécanisme d'apprentissage par mémorisation des conflits, permettant l'exploitation de l'effort de résolution des CSPs statiques successifs.

Des expérimentations menées dans un cadre industriel et sur une problématique réelle montrent l'efficacité de la nouvelle heuristique et du mécanisme d'apprentissage.

Abstract

In this paper, we propose a new dynamic variable ordering heuristics, $dom/wdeg_{cbj}$, for a hybrid method which combining a look-ahead and a look-back schema, implemented respectively by *Forward-Checking* (FC) and *Conflict Directed Backjumping* (CBJ). This heuristics is a conflict-directed variable ordering heuristics and inspired by the heuristics *dom/wdeg* proposed by Boussemart[11] for a look-ahead methods. This new heuristics allows to reduce thrashing phenomenon by first instantiating variables that have frequently participated in conflict situations. Such information is recorded by associating a weight to each variable. During a conflict situation that requiring a backtrack return, the weight of the current variable and the variable which is responsible for the conflict situation (identified by CBJ) are increased. We also propose, within a context of a Dynamic CSP of T CSPs, a learning mechanism by conflict recording, allowing to exploit the resolution efforts of the successive static CSPs.

Experiments led in an industrial context and on a real problem show the efficiency of the new heuristics and the learning mechanism.

1 Introduction

Depuis l'avènement des algorithmes de recherche arborescente pour la résolution des Problèmes de Satisfaction de Contraintes (CSPs), ces derniers n'ont eu de cesse d'évoluer au travers de nombreuses améliorations. Ces améliorations permettent de limiter l'explosion combinatoire inhérente à ce type d'approches. Elles portent principalement sur des mécanismes mis en œuvre durant la phase de progression « look-ahead »[9], tel que l'heuristique de choix de variable (de valeur) et le mécanisme de filtrage, et durant la phase de régression « look-back »[9] tel que l'analyse

des conflits.

Durant les années 90, l'algorithme prospectif Forward-Checking (FC) [16] associé à la méthode rétrospective Conflict Directed Backjumping (CBJ) [20, 22, 14], et la méthode prospective Maintien de l'Arc Coherence (MAC) [13, 21], étaient considérés comme étant les algorithmes les plus efficaces pour résoudre les instances CSPs. Cependant, la méthode MAC avait un avantage significatif lors de la résolution d'instances difficiles de grande taille [4, 7]. Cet avantage a été remis en question par [2, 15, 19], qui proposent une amélioration du mécanisme rétrospectif en associant une explication non pas à chaque variable mais à chaque valeur, donnant ainsi une capacité de retour arrière plus importante. De telles méthodes rétrospectives semblent surclasser la méthode prospective MAC pour des instances difficiles de grande taille.

Néanmoins, les techniques prospectives ont repris un certain avantage depuis l'introduction, par Boussemart, d'une nouvelle heuristique de choix de variable dirigée par les conflits [11]. Elle permet de traiter plus efficacement les problèmes structurés et non structurés en orientant la recherche vers les parties difficiles ou inconsistantes du problème. L'idée est d'associer un poids à chaque contrainte et d'incrémenter ce poids à chaque fois que cette dernière est violée pendant la recherche. Ces poids permettent ensuite de définir un degré pondéré *wdeg* pour chaque variable. Combiné à la taille du domaine courant *dom*, le degré pondéré *wdeg* permet de définir l'heuristique de choix de variable *dom/wdeg*, qui s'apparente à de la mémorisation de *nogoods*, une forme d'apprentissage à partir des conflits. L'heuristique *dom/wdeg* réussit à combiner avantageusement l'aspect prospectif et rétrospectif des algorithmes de recherche arborescente. Elle permet de limiter le phénomène de trashing (redécouverte des mêmes inconsistances) et est une alternative simple aux techniques de retours arrières intelligents[6].

À notre connaissance, l'étude de l'heuristique *dom/wdeg* n'a été menée que dans le cadre de méthodes prospectives liées à un schéma de recherche de type retour arrière chronologique (backtrack), telles que *FC* et *MAC*. Dans cet article nous proposons une nouvelle heuristique dirigée par les conflits $dom/wdeg_{cbj}$. $dom/wdeg_{cbj}$ est inspirée de *dom/wdeg* et s'applique à une méthode hybride alliant un schéma prospectif et un schéma rétrospectif, mis en œuvre respectivement par le Forward-Checking (FC) et le Conflict Directed Backjumping (CBJ) : *FC-CBJ*. Elle exploite la capacité de *CBJ* à identifier la variable source de l'échec. Pour un coût de mise en œuvre limité, cette nouvelle heuristique permet des gains significatifs sur notre exemple d'application, que ce soit en nombre d'affectation (de nœuds) ou en temps CPU. Pour des raisons pratiques, l'étude de l'heuristique $dom/wdeg_{cbj}$ est limitée au cadre des CSPs binaires.

Nous proposons également, dans le cadre des CSPs Dynamiques (DCSP), un mécanisme d'apprentissage permettant d'exploiter l'effort de résolution des CSPs statiques successifs. Ce mécanisme d'apprentissage consiste à mémoriser les poids entre les CPSs successifs $P_{(0)}, P_{(1)}, \cdots, P_{(N)}$. L'idée principale est d'initialiser les poids du CPS $P_{(i)}$ par ceux du CPS $P_{(i-1)}$.

Les deux contributions académiques de cet article, l'heuristique dynamique de choix de variable et le mécanisme d'apprentissage pour les DCSP, ont été identifiées dans un context industriel, durant l'amélioration d'une méthode existante permettant de valider le design de la matrice de routage d'une charge utile télécom. Cette validation consiste à évaluer la tolérance d'une matrice à un nombre fixé de pannes de tubes amplificateurs TWTA (pour Travelling Wave Tube Amplifier). La propriété de tolérance est assurée par l'existence d'un mécanisme de redondance basé sur un ensemble R de tubes TWTA redondant et une matrice de routage permettant d'accéder aux tubes TWTA nominaux et redondant. Dans le cas de R pannes de TWTAs nominaux (utilisés dans des conditions nominales), la matrice de routage permet d'accéder aux R TWTAs redondant en considérant la configuration courante de la charge utile. Afin de valider le design de la matrice de routage et de fournir des statistiques sur les cas de pannes pour lesquels il n'y a pas d'accès possible aux TWTAs redondant via la matrice de routage, le schéma de validation choisie consiste en une énumération exhaustive de toutes les combinaisons, où une combinaison est définie par une configuration de la charge utile et un ensemble de R TWTAs en pannes. Une combinaison pour laquelle il n'y a pas d'accès possible aux TWTAs redondant via la matrice de routage, est dite *non valide*. Ces statistiques sont ensuite utilisés par les ingénieurs afin de modifier le design si besoin. Pour une matrice donnée, le nombre N de combinaisons peut être égale à plusieurs millions dans le meilleur des cas, voir plusieurs milliards dans le pire. La validation quant à elle peut nécessiter plusieurs heures dans le meilleur des cas, voir plusieurs jours dans le pire.

Soit $P_{(i)}$ un CSP binaire associé à une combinaison i, avec des contraintes de compatibilité entre les valeurs. Ainsi, valider le design d'une matrice de routage consiste à résoudre la séquence $P_{(0)}, P_{(1)}, \cdots, P_{(N)}$ de N CSPs, où chacun est le résultat d'un certain nombre de changements appliqués au précédent, ce qui correspond à la definition d'un Problème Dynamique de Satisfaction de Contraintes (DCSP). Ces changements sont des ajouts et/ou suppressions de valeurs, et activations et désactivations de variables (et/ou

de contraintes), qui correspondent respectivement à l'énumération des sous-combinaisons de R pannes de TWTAs et à l'énumération des sous-combinaisons de configuration de charge utile. Pour chaque configuration, T sous-combinaisons de R pannes de TWTAs sont générées, ainsi $N = PC * T$, où PC est le nombre de configurations de charge utile. Une combinaison dont le CSP associé admet au moins une solution est dite *valide*, *non-valide* dans le cas contraire.

La particularité de cette sequence de CSPs tient dans le fait que tous les T CPSs, une variable est activée et une autre est désactivée, et entre les CSPs successifs ayant les mêmes variables actives, des changements de valeurs sont effectués. Ainsi, deux CSPs séparés de $T-1$ CSPs ne diffèrent que d'une variable et de quelques valeurs. Il en résulte une forte similitude entre les CSPs successifs. Pour des raisons de confidentialité, nous ne somme pas autorisé à donner plus de details sur la modélisation. Cependant, cela ne nuira pas à la comprehension de l'article.

La méthode utilisée pour résoudre chaque CSP est la méthode hybride *FC-CBJ* associée à de l'Arc Cohérence (AC3.2bit [17]) en phase de prétraitement et à la nouvelle heuristique dirigée par les conflits, $dom/wdeg_{cbj}$, qui constitue notre premiere contribution. La seconde contribution est le mécanisme d'apprentissage des conflits entre les CSPs successifs. Cette combinaison de méthodes et de mécanismes fournit les meilleurs performances sur notre problématique de validation, notamment face à la méthode prospective Maintien de l'Arc Coherence (MAC).

Cet article est organisé comme suit. Dans un premier temps, nous donnons quelques définitions classiques dans le domaine des CSPs. Par la suite, nous décrivons plus en detail l'heuristique *dom/wdeg* proposée dans [11] et la nouvelle heuristique $dom/wdeg_{cbj}$ que nous proposons. Suite à quoi, nous présentons le mécanisme d'apprentissage appliqué à un DCSP binaire de T CSPs. Nous présentons ensuite les résultats d'expérimentations menées sur des instances CSPs structurées issues de la problématique industrielle de validation. Nous terminerons avec une conclusion et des perspetives.

2 Préliminaires

Dans cette section nous introduisons quelques définitions et notations utilisées dans la suite de l'article. Ces définitions et notations sont issues de [11].

Un Problème de Satisfaction de Contrainte est défini par un ensemble de variables prenant leurs valeurs dans un domaine et un ensemble de contraintes qui conditionnent les valeurs que pourront prendre les variables. Il est généralement présenté sous la forme d'un réseau de contraintes (V, C).

Définition 1 *Un réseau de contraintes est un couple (V, C) où :*

- *$V = \{V_1, \cdots, V_n\}$ est un ensemble fini de n variables. À chaque variable V_i correspond un domaine $dom(V_i)$ représentant l'ensemble des valeurs pouvant lui être affectée.*
- *$C = \{C_1, \cdots, C_m\}$ est un ensemble fini de m contraintes. Une contrainte $C_j \in C$ est associée à une relation $rel(C_j)$ représentant l'ensemble des tuples autorisés pour les variables $vars(C_j)$ liées par la contrainte C_j.*

Une contrainte C_j lie une variable V_i si et seulement si $V_i \in vars(C_j)$. Le nombre $|vars(C_j)|$ de variables liées par une contrainte C_j définit l'arité de celle-ci. Une contrainte binaire lie deux variables (arité égale à deux). En revanche, le nombre de contraintes liant une variable V_i représente le degré de cette dernière.

L'affectation d'une valeur à une variable est appelée instantiation (ou assignation). Soit $s = (v_1, v_2, \cdots, v_p)$ une instantiation partielle où p variable sont affectées, avec $p = |s| \leq |V|$ et v_i la valeur affectée à la variable V_i. Si $p = |V|$, s est appelée instantiation totale. Une solution est une instantiation totale de sorte que chaque contrainte soit satisfaite.

Un réseau de contraintes est aussi appelé instance CSP. Par abus de langage, CSP et instance CSP sont parfois confondus. Sans perte de généralité, dans cet article, L'ensemble $vars(C_j)$ des variables liées à la contrainte C_j est considéré comme étant ordonné. Il est alors possible d'obtenir la position $pos(V_i, C_j)$ d'une variable V_i dans $vars(C_j)$. Un tuple t est dit autorisé par une contrainte C_j si et seulement si $t \in rel(C)$. Un tuple t est dit support de l'instantiation (V_i, v) dans C_j si t est autorisé par C_j et tel que $t[pos(V_i, C_j)] = v$. (V_i, v) est dite consistant par rapport à C_j si et seulement si il existe un support de (V_i, v) dans C_j. L'opération consistant à déterminer si un tuplet est support d'une contrainte est appelé test de consistance

Définition 2 *Soient P une instance CSP, C_j une contrainte de P, V_i une variable de $vars(C_j)$ et v une valeur de $dom(V_i)$. (V_i, v) est dit consistant par rapport à C_j si et seulement si il existe un support de (V_i, v) dans C_j. C_j est dit arc-consistant si et seulement si pour toute variable V_i de $vars(C_j)$ et pour toute valeur v de $dom(V_i)$, (V_i, v) est consistant par rapport à C_j. P est arc-consistant si et seulement si toute contrainte de P est arc-consistante.*

La résolution d'un Problème de Satisfaction de Contrainte est la tâche de trouver une solution à une

instance CSP. Ce problème est NP-complet. Une instance CSP est dite satisfiable si et seulement si le réseau de contraintes associé admet au moins une solution. Dans le cas contraire, cette instance est dite non-satisfiable. Résoudre un CSP consiste alors, soit à déterminer sa non-satisfiabilité, soit à trouver au moins une solution.

3 Pondération des contraintes à partir des conflits pour une méthode prospective

L'un des points d'amélioration d'un algorithme de recherche arborescente de type retour arrière est l'ordre d'instantiation des variables, un ordre considéré comme crucial depuis l'avènement de ce type d'algorithme. Les heuristiques généralement utilisées sont *dom* [16], *dom/deg* [4], *dom/ddeg* [4, 24], $dom \oplus deg$ [12] et $dom \oplus ddeg$ [23, 5]. Ces heuristiques exploitent un certain nombre d'informations sur l'état courant de la recherche, telles que la taille courante des domaines (*dom*) ou le degré (courant) des variables (*deg* dans le cas statique et *ddeg* dans le cas dynamique). Aucune information sur l'état passé n'est exploitée.

En 2004, [11] proposent, dans un cadre général, celui des CSPs n-aire, une heuristique de choix de variable originale, *wdeg* (pour weighted degree), dirigée par les conflits. Cette heuristique dynamique est basée sur un poids *weight* associé à chaque contrainte du problème. Pour une contrainte C_j, le poids *weight* est incrémenté de un à chaque conflit impliquant cette dernière. L'évaluation du degré pondéré $\alpha_{wdeg}(V_i)$ d'une variable V_i est alors égale à la somme des poids des contraintes liant V_i et au moins une seconde variable non instantiée ; plus formellement :

- $\alpha_{wdeg}(V_i) = \sum_{V_i \in vars(C_j) \wedge |FutVars(C_j)|>1} weight[C_j]$

, où $| \; FutVars(C_j) \; |$ représente le nombre de variables non instantiées dans $vars(C_j)$. Dans la suite de l'article, le degré pondéré α_{wdeg} est noté *wdeg*. Dans le même article, les auteurs proposent l'heuristique *dom/wdeg*, qui consiste à sélectionner prioritairement la variable avec le plus petit rapport : taille courante du domaine de la variable, *dom*, sur le degré pondéré de cette dernière, *wdeg*. L'heuristique *dom/wdeg* compense l'absence d'un schéma rétrospectif (analyse des conflits) par un apprentissage lui permettant d'orienter la recherche vers les parties difficiles ou inconsistantes du problème.

Les auteurs ont proposé cette heuristique dans le cadre de méthodes prospectives tel que le *Forward-Checking* [16] (FC) et le *Maintien de l'Arc Coherence* [13, 21] (MAC). Ces méthodes mettent en œuvre des mécanismes de filtrage permettant de modifier le problème selon l'assignation réalisée, ceci grâce à des révisions successives des variables de manière à éliminer les valeurs devenues inconsistantes. Cette révision correspond par exemple à l'utilisation d'un filtrage gros grain (orienté arc pour cet exemple). Soit Q l'ensemble des arcs à reviser, un arc étant défini comme un couple *(Contrainte, Variable)*. La révision de l'arc (C_j, V_i) consiste à éliminer les valeurs de $dom(V_i)$ devenues inconsistantes avec C_j.

Algorithme 1 filtrer(Q : ensemble d'arcs) :booleen

```
1:  tantque Q ≠ ∅ faire
2:     Sélectionner et éliminer (C_j, V_i) de Q
3:     si reviser(C_j, V_i) alors
4:        si dom(V_i) = ∅ alors
5:           weight[C_j]++ // incrémentation du compteur de la contrainte
6:           retourner ECHEC
7:        sinon
8:           mettre à jour Q // en fonction de l'algorithme de filtrage
9:        finsi
10:    finsi
11: fin tantque
12: retourner SUCCES
```

La fonction *filter*, qui prend en paramètre l'ensemble Q, permet de reviser un certain nombre d'arcs itérativement (l'initialisation de Q est spécifique à FC et à MAC). Lorsqu'une impasse ($dom(V_i) = \emptyset$) est rencontrée lors d'une révision effective de l'arc (C_j, V_i), le poids *weight* associé à la contrainte C_j est incrémenté de un. Le filtrage se termine alors sur un échec. Dans le cas contraire, l'ensemble Q est mis à jour. La révision est effectuée par la fonction *reviser*.

Algorithme 2 reviser(C_j : contrainte, V_i : Variable) :booleen

```
nbElements ← |dom(V_i)|
pour chaque v ∈ dom(V_i) faire
   si rechercherSupport(C_j, V_i, v) = false alors
      retirer v de dom(V_i)
   finsi
fin pour
retourner nbElements ≠ |dom(V_i)|
```

Les auteurs ont montré l'efficacité d'une telle heuristique face aux heuristiques classiques sur un certain nombre de problèmes académiques structurés et non structurés. Cependant, cette heuristique n'a été présentée que pour des méthodes prospectives, dont le schéma d'exploration est du type retour arrière chronologique (backtrack). On peut donc s'interroger sur l'apport de la transposition d'une telle heuristique

pour des méthodes rétrospectives, voir hybrides, tel que la méthode hybride *FC-CBJ* alliant un schéma prospectif et un schéma rétrospectif, mis en œuvre respectivement par le *Forward-Checking* et le *Conflict Directed Backjumping*.

4 Heuristique dirigée par les conflits pour une méthode hybride : FC-CBJ

La méthode hybride *FC-CBJ* exploite la capacité de filtrage à moindre coût de la méthode prospective *Forward-Checking* et la capacité de retour arrière de la méthode rétrospective *Conflict Directed Backjumping* qui permet d'identifier la variable source de l'échec. Dans [11], l'heuristique *dom/wdeg* ne dispose pas de cette information (la variable source de l'échec) car elle est appliquée à un schéma de recherche de type retour arrière chronologique (backtrack). Appliquer l'heuristique *dom/wdeg* à la méthode *FC-CBJ* sans prendre en compte la capacité de cette dernière à identifier les variables sources de l'échec, risque d'orienter la recherche vers des parties non nécessairement difficiles ou inconsistantes du problème.

Nous proposons d'enrichir la phase d'apprentissage de l'heuristique *dom/wdeg* en redéfinissant le poids *weight* afin d'exploiter la capacité de la méthode retrospective *CBJ* à identifier la source de l'échec. Il en résulte une nouvelle heuristique dirigée par les conflits, $dom/wdeg_{cbj}$. Pour des raisons pratiques, l'étude de l'heuristique $dom/wdeg_{cbj}$ est limitée au cadre des CSPs binaires.

Tout d'abord, il est important de remarquer que dans un cadre binaire, il y a équivalence entre associer un compteur *weight* à chaque contrainte ou à chaque variable. Pour cela il suffit de remplacer la ligne 5 de l'algorithme 1 par les deux lignes suivantes :

- $weight[V_i][secondVar(C_j, V_i)] + +$
- $weight[secondVar(C_j, V_i)][V_i] + +$

C_j est la contrainte à l'origine d'une impasse lors de la révision de l'arc (V_i, C_j), et $secondVar(C_j, V_i)$ une fonction retournant la seconde variable liée par la contrainte C_j, la première étant la variable V_i (voir l'algorithme 3). Incrémenter le poids *weight* associé à une contrainte C_j d'arité égale à deux revient donc à incrémenter les poids *weight* associés aux deux variables liées par C_j. Le degré pondéré d'une variable V_i s'exprime alors de la façon suivante :

- $\alpha_{wdeg}(V_i) = \sum_{V_k \in V, V_k \in FutVars(V)} weight[V_i][V_k]$

Où $\mid FutVars(V) \mid$ représente l'ensemble des variables non instantiées appartenant à V.

Algorithme 3 filtrer(Q : ensemble d'arcs) :booleen

1: **tantque** $Q \neq \emptyset$ **faire**
2: Sélectionner et éliminer (C_j, V_i) de Q
3: **si** reviser(C_j, V_i) **alors**
4: **si** $dom(V_i) = \emptyset$ **alors**
5: $weight[V_i][secondVar(C_j, V_i)] + +$
6: $weight[secondVar(C_j, V_i)][V_i] + +$
7: retourner ECHEC
8: **sinon**
9: mettre à jour Q
10: **finsi**
11: **finsi**
12: **fin tantque**
13: retourner SUCCES

Comme dit précédemment, afin d'exploiter la capacité du *CBJ* à identifier la source de l'échec, nous proposons d'enrichir la phase d'apprentissage de l'heuristique *dom/wdeg*. Pour cela, nous redéfinissons le mécanisme d'incrémentation du poids *weight*, qui est à présent nommé $weight_{cbj}$. Durant une recherche arborescente de type *FC-CBJ*, lors de l'échec de l'instantiation de la variable courante V_{curr} ($dom(V_{curr}) = \emptyset$ à cause des révisions successives de FC), la méthode rétrospective *CBJ* permet d'identifier la variable V_{jump} dont l'affectation courante est responsable de l'échec. La variable V_{jump} doit alors changer de valeur afin de débloquer la situation. L'idée est alors de conserver l'information d'échec entre les deux variables en incrémentant les poids $weight_{cbj}$ associés aux variables V_{curr} et V_{jump}. Le poids $weight_{cbj}$ nous permet ensuite d'évaluer le degré pondéré $\alpha_{wdeg_{cbj}}(V_i)$ de la variable V_i et ainsi de définir la nouvelle heuristique dynamique $dom/wdeg_{cbj}$, qui consiste à sélectionner prioritairement la variable avec le plus petit rapport : taille courante du domaine de la variable, *dom*, sur le degré pondéré $wdeg_{cbj}$ de cette dernière.

Dans le but de mettre en évidence l'intérêt de la redéfinition du mécanisme d'incrémentation du poids *weight* lors de l'adaptation de l'heuristique *dom/wdeg* à la méthode *FC-CBJ*, nous avions choisi d'associer les poids *weight* (et $weight_{cbj}$) aux variables (ce qui est équivalent à les associer aux contraintes dans le cas binaire). Cependant, il est possible d'adopter la définition de [11]. Pour cela, les explications d'échec doivent être maintenues sous forme d'ensemble de contraintes menant au conflit, ce qui permet de maintenir la notion de poids associé à chaque contrainte. En cas de conflit, il suffit alors d'incrémenter les poids des contraintes présentent dans le conflit. Ainsi, le degré pondéré d'une variable aurait la même définition que dans [11] et l'heuristique $dom/wdeg_{cbj}$ devient alors applicable dans un cadre n-aires.

5 Mémorisation de conflits pour un CSP Dynamique de T CSPs

Un CSP Dynamique (DCSP) a été introduit par [10] comme une suite de CSPs statiques $P_{(0)}, P_{(1)}, \cdots, P_{(N)}$, chacun résultant du précédent par des opérations sur les valeurs (ajout et/ou suppression). Dans [18], un DCSP est une généralisation d'un CSP [25], qui est défini par un ensemble de variables, un état d'activation des variables, un domaine pour chaque variable, et par deux types de contraintes : contraintes de compatibilité qui régissent l'association variable-valeur, et contraintes d'activation qui permettent d'activer ou de désactiver une variable selon l'assignation des autres variables. Une contrainte d'activation est de la forme : Si la valeur v est assignée à la variable V_i, alors $V_j, V_g, \cdot$ deviennent actives. Le terme *conditionnelle* serait peut être plus approprié.

Dans le cadre de la résolution de Problème Dynamique de Satisfaction de Contraintes[10, 18], de nombreux travaux ont été menés sur l'exploitation de l'effort de résolution des CSPs statiques successifs, notamment sur la réutilisation d'une solution précédemment calculée [1, 26], l'exploitation des explications de la non faisabilité [22] et l'adaptation des algorithmes d'arc consistance [3, 8]. Dans la même optique, nous proposons un mécanisme d'apprentissage par la mémorisation des conflits. En fonction de la similitude entre les CSPs successifs, une fois la recherche arborescente menée à terme, avec soit une preuve de la non-satisfiabilité, soit une preuve de satisfiabilité pour une instance CSP $P_{(i)}$, les poids $weight_{cbj}$ (or $weight$) permettent de mettre en évidence les parties difficiles ou inconsistantes de $P_{(i)}$, et potentiellement de $P_{(i+1)}$. Ainsi, durant la résolution de $P_{(i+1)}$, qui résulte d'un certain nombre de changements dans $P_{(i)}$, il serait intéressant de traiter en priorité les parties difficiles ou inconsistantes identifiées durant la résolution de $P_{(i)}$. Ceci permettrait, selon le principe *fail-first*, soit de détecter la non-satisfiabilité de l'instance plus tôt, soit d'accélérer la résolution en traitant en priorité les parties difficiles du problème (simplification des sous-problèmes inhérents). Dans la suite de cette section, ce qui est appliqué au poids $weight_{cbj}$ l'est également au poids $weight$.

Le mécanisme d'apprentissage est proposé dans le cadre d'un Problème Dynamique de Satisfaction de Contraintes (DCSP) de T CSPs, qui est une combinaison des deux définitions précédentes. Un DCSP de T CSPs est une séquence $P_{(0)}, P_{(1)}, \cdots, P_{(N)}$ de CSPs, chacun résultant du précédent par un certain nombre de changements. Ces changements sont des ajouts et/ou suppressions de valeurs, et activations et désactivations de variables. La particularité de cette sequence de CSPs tient dans le fait que tous les T CPSs, une variable est activée alors qu'une autre est désactivée, et entre les CSPs successifs ayant les mêmes variables actives, des changements de valeurs sont effectués. Ainsi, deux CSPs séparés de $T-1$ CSPs ne diffèrent que d'une variable et de quelques valeurs. Il en résulte une forte similitude entre les CSPs successifs.

Le mécanisme d'apprentissage consiste à initialiser le poids $weight_{cbj}$ de chaque variable du CSP $P_{(i+1)}$ par celui du CSP $P(i)$ ayant les mêmes variables. Quand deux CSPs successifs different d'une variable (et de quelques valeurs), les poids $weight_{cbj}$ sont réinitialisés à 1, ce qui permet de préserver le caractère discriminant des poids $weight_{cbj}$. Finalement, les poids $weight_{cbj}$ sont réinitialisés tout les T CSPs, d'où un DCSP de T CSPs. Il est important de noter que ce mécanisme est applicable à un DCSP quelconque. Dans la suite de l'article, l'association de la méthode *FC-CBJ*, de l'heuristique $dom/wdeg_{cbj}$ et du mécanisme d'apprentissage est notée $dom/wdeg^{T}_{cbj}$.

6 Exemple d'application et résultats

Comme mentionné précédemment, les contributions académiques de cet article on été identifiées dans un context industriel, durant l'amélioration d'une méthode existante permettant de valider le design de la matrice de routage d'une charge utile télécom. Cette validation consiste à résoudre une séquence $P_{(0)}, P_{(1)}, \cdots, P_{(N)}$ de CSPs binaires. Cette séquence résulte d'une énumération exhaustive de toutes les combinaisons, où une combinaison est définie par une configuration de la charge utile et un ensemble de R TWTAs en pannes. La séquence de CSPs binaires respecte la définition d'un DCSP de T CSPs binaire.

Afin de mettre en évidence l'intérêt pratique de ces contributions, nous avons conduit des expérimentations sur la problématique de validation (sur une machine Pentium Dual Core 2GHz 2Go , sous Windows XP). La nouvelle heuristique $dom/wdeg_{cbj}$ que nous proposons et le mécanisme d'apprentissage sont évalués selon le nombre de nœuds (#asgs) et le temps CPU de résolution (cpu, $hh : mm : ss$).

Dans un premier temps, nous comparons l'heuristique $dom/wdeg$ [11] et la nouvelle heuristique $dom/wdeg_{cbj}$ que nous proposons, toutes deux appliquées à la méthode de résolution précédemment décrite dans la section 1. Dans un second temps, nous évaluons l'apport de notre mécanisme d'apprentissage lors de la résolution des CSPs binaires successifs, dans le cadre d'un DCSP de T CSPs, où chaque CSP est résolu par la méthode décrite dans la section 1.

Avant de continuer, nous devons apporter une precision supplémentaire concernant le processus de va-

lidation. La problématique de validation est sujette à une contrainte supplémentaire qui limite le nombre de valeurs par variable. À la fin d'une validation, si des combinaisons non-valides ont été détectées, le processus de validation est relancé pour ces seules combinaisons, mais en incrémentant la borne supérieure de cette contrainte. Ainsi les nouvelles valeurs introduites permettant de diminuer le nombre de combinaisons non-valides. Le processus de validation est relancer tant que des combinaisons non-valides sont détectées, sous condition que la borne supérieure de la contrainte n'atteigne pas sa valeur maximale. Il faut noter que le processus de validation peut être lancé avec une borne supérieure égale à sa valeur maximale.

Les expérimentations sont menées sur trois types de DCSP, A, B et C, chacun correspondant à un design particulier de matrice de rouage. Soit B_i et C_i, avec $i = 1 \cdots 4$, des variantes, respectivement des DCSP B et C. Ces variantes résultent de l'ajout de valeurs inhérent à l'incrémentation de la borne supérieure de la contrainte précédemment exposée. Le tableau 1 illustre, par DCSP, le nombre de combinaisons, le nombre de combinaisons non-valides et les caractéristiques (nombre de variables, taille moyenne des domaines, un ordre de grandeur du nombre de contraintes) des CSPs associés aux combinaisons. Le DCSP B_1 a 10 829 485 combinaisons non-valides parmi 25 603 600 combinaisons. À la fin de la validation de B_1, les combinaisons non valides identifiées (10 829 485) vont composer le DCSP B_2, auquel des valeurs supplémentaires sont ajoutées du fait de l'incrémentation de la borne supérieure de la contrainte limitant le nombre de valeurs par variable. De la même manière, B_3 résulte de B_2, C_2 de C_1 et C_3 de C_2. Les DCSP B_4 et C_4 corresponde quant à eux, à une validation avec une borne supérieure égale à sa valeur maximale.

Le tableau 2 présente des statistiques en nombre d'assignation (#asgs) et de temps de validation (cpu) permettant de départager $dom/wdeg_{cbj}$ et $dom/wdeg$. On remarque, que pour l'ensemble des DCSPs, regroupant 86 126 235 de CSPs dont 26 594 967 non-satisfiables, l'heuristique $dom/wdeg_{cbj}$, que nous proposons, donne de meilleurs résultats en terme de nombre d'assignations et de temps cpu que l'heuristique $dom/wdeg$. La redéfinition des poids $weight$ a permis de d'exploiter, plus efficacement, la capacité de la méthode retrospective *CBJ* à identifier la source du conflit. Pour un coût d'implémentation limité, le nombre d'assignations et le temps cpu ont été respectivement réduits de 1,43% à 21,51% et de 0,66% à 17,61%. Ainsi, au lieu d'appliquer $dom/wdeg$ à al méthode hybride *FC-CBJ*, il est plus intéressant d'appliquer son extension, l'heuristique $dom/wdeg_{cbj}$.

Le tableau 3 illustre des statistiques en nombre d'assignations (#asgs) et de temps de validation (cpu) permettant d'évaluer l'apport du mécanisme d'apprentissage. On remarque une réduction significative de ces deux paramètres pour les DCSPs A_1, B_1, C_1, C_2 et C_4. Par exemple, le temps de validation du DCSP C_4 est égale à environ 13 heures avec le mécanisme d'apprentissage, alors qu'il est d'environ deux jours et demi sans le mécanisme d'apprentissage. Il est également important de remarquer que la validation des DCSPs B_3, B_4 et C_3 voient leurs temps cpu augmenter (au maximum de 9%) avec le mécanisme d'apprentissage. Néanmoins, ce temps reste inférieur à celui obtenu avec l'heuristique $dom/wdeg$ sans mémorisation. Ainsi, avec un coût d'implementation très faible, le mécanisme d'apprentissage offre des améliorations significatives sur notre exemple d'application.

7 Conclusion

Dans cet article, nous avons introduit une nouvelle heuristique dynamique de choix de variable dirigée par les conflits, l'heuristique $dom/wdeg_{cbj}$, appliquée à une méthode hybride *FC-CBJ*, alliant un schéma prospectif et un schéma rétrospectif, mis en œuvre respectivement par le *Forward-Checking* et le *Conflict Directed Backjumping*. Nous avons également proposé un mécanisme d'apprentissage permettant d'exploiter l'effort de résolution des CSPs statiques successifs dans le cadre d'un DCSP de T CSPs. Ce mécanisme, simple et peu coûteux, repose sur la mémorisation des compteurs $weight_{cbj}$ entre les CSPs successifs. L'heuristique $dom/wdeg_{cbj}$ a été fortement inspirée par l'heuristique $dom/wdeg$ [11] de Boussemart et al. Néanmoins, $dom/wdeg_{cbj}$ se distingue de $dom/wdeg$ par un apprentissage par les conflits plus adapté à la méthode rétrospective *CBJ*, ce qui lui permet de la surclasser en terme de nombre de nœuds et de temps CPU sur notre exemple d'application. Les expérimentations menées sur cet exemple d'application ont permis de mettre en évidence l'apport significatif du mécanisme d'apprentissage, notamment sur le design C4 qui est actuellement utilisé pour l'élaboration des nouvelles matrices de routage.

L'heuristique $dom/wdeg_{cbj}$ et le mécanisme d'apprentissage ont été proposé dans un cadre bien particulier, celui des CSPs binaires et des DCSP de T CSPs. Néanmoins, comme dit précédemment, leurs transcriptions dans un cadre plus général, celui des CSPs n-aire et des DCSPs quelconque est possible. De plus, le mécanisme d'apprentissage peut être étendu à d'autres méthodes de résolution, autre que la méthode hybride *FC-CBJ*.

Remerciements

Nous tenons à remercier Christophe Lecoutre pour avoir mené des expérimentations sur un échantillon de combinaisons et de nous avoir orienté vers l'heuristique *dom/wdeg*.

Références

[1] J. Maloney B. N. Freeman-Benson and A. Borning. An incremental constraint solver. *Communications of the ACM*, 33(1) :54–63, 1990.

[2] F. Bacchus. Extending forward checking. In *Proceedings of the CP'00*, pages 35–51, 2000.

[3] C. Bessière. Arc-Consistency in Dynamic Constraint Satisfaction Problems. In *Proceeding of The Ninth National Conference on Artificial Intelligence*, pages 221–226, 1991.

[4] C. Bessiere and J. Regin. MAC and combined heuristics : two reasons to forsake FC (and CBJ ?) on hard problems. In *Proceedings of the CP'96*, pages 61–75, 1996.

[5] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22 :251–256, 1979.

[6] F. Boussemart C. Lecoutre and F. Hemery. Techniques de retour-arrière intelligent versus heuristiques dirigées par les conflits. In *Proceedings of JNPC'04*, 2004.

[7] A. Chmeiss and L. Saïs. De FC à MAC : un algorithme paramétrable pour la résolution des CSP. pages 267–276, Lens, 2005.

[8] R. Debruyne. Les algorithmes d'arc-consistence dans les csp dynamiques. *Revue d'intelligence artificielle*, 9(3) :239–267, 1995.

[9] R. Dechter and D. Frost. Backjump-based backtracking for constraint satisfaction problems. *Artificial Intelligence*, 136 :147–188, 2002.

[10] R. Detcher and A. Detcher. Belief maintenance in dynamic constraint networks. In *Proceedings of the 7th National Conference on Artifical intelligence*, pages 37–42, St. Paul, MN, USA, 1988.

[11] C. Lecoutre F. Boussemart, F. Hemery and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of the ECAI'O4*, pages 146–150, 2004.

[12] D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of IJCAI'95*, pages 572–578, 1995.

[13] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical report CMU-CS-79-124, Carnegie-Mellon, 1979.

[14] I. P. Gent and I. P. Underwood. The logic of search algorithms : Theory and applications. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP'97)*, pages 77–91, 1997.

[15] M. Ginsberg. Dynamic backtracking. *Artificial Intelligence*, 1 :25–46, 1993.

[16] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14 :263–313, 1980.

[17] C. Lecoutre and J. Vion. Enforcing arc consistency using bitwise operations. *Constraint Programming Letters(CPL)*, 2 :21–35, 2008.

[18] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artifical intelligence*, pages 25–32, 1990.

[19] R. Debruyne N. Jussien and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Proceedings of the CP'00*, pages 249–261, 2000.

[20] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3) :268–299, August 1993.

[21] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of the PPCPA'94*, Seattle WA, 1994.

[22] T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *IJAIT*, 3(2) :187–207, 1994.

[23] B.M. Smith. The brelaz heuristic and optimal static orderings. In *Proceedings of CP'99*, pages 405–418, Alexandria, VA, 1999.

[24] B.M. Smith and S.A. Grant. Trying harder to fail first. In *Proceedings of ECAI'98*, pages 249–253, Brighton, UK, 1998.

[25] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press Limited, Sandiego, 1993.

[26] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proceeding of the Twelfth National Conference on Artificial Intelligence*, pages 307–312, 1994.

DCSPs	Nombre de combinaisons		nbVar	nbVal	nbContraintes
	Total	Invalide		(moyenne)	(≈)
A_1	11 625 120	240	22	8	≈ 900
B_1	25 603 600	10 829 485	21	10	≈ 2400
B_2	10 829 485	8 695 364	21	14	≈ 8000
B_3	8 695 364	3 126 532	21	22	≈ 28000
B_4	25 603 600	3 126 532	21	22	≈ 28000
C_1	7 312 032	475 768	25	11	≈ 4500
C_2	475 768	294 354	25	19	≈ 21000
C_3	294 354	23 466	25	34	≈ 87000
C_4	7 312 032	23 466	25	34	≈ 87000

TABLE 1 – Données des diffèrent DCSPs

DCSPs		$dom/wdeg$	$dom/wdeg_{cbj}$	$dom/wdeg_{cbj}$ vs $dom/wdeg$
A_1	#asgs	556 791 037	548 839 044	**-1,43%**
	cpu	00 :58 :14	00 :57 :51	**-0,66%**
B_1	#asgs	1 854 403 849	1 628 697 906	**-12,17%**
	cpu	03 :45 :21	03 :30 :57	**-6,39%**
B_2	#asgs	8 444 489 777	7 644 525 648	**-9,47%**
	cpu	12 :03 :07	11 :02 :34	**-8,37%**
B_3	#asgs	19 335 093 642	17 250 317 028	**-10,78%**
	cpu	28 :41 :01	26 :05 :21	**-9,05%**
B_4	#asgs	21 059 047 647	18 860 453 983	**-10,44%**
	cpu	34 :11 :17	31 :24 :56	**-8,11%**
C_1	#asgs	745 606 878	696 215 827	**-6,62%**
	cpu	02 :02 :07	01 :58 :34	**-2,91%**
C_2	#asgs	2 696 933 390	2 116 805 967	**-21,51%**
	cpu	04 :03 :51	03 :20 :54	**-17,61%**
C_3	#asgs	5 136 698 506	4 776 536 475	**-7,01%**
	cpu	09 :26 :42	08 :54 :25	**-5,70%**
C_4	#asgs	22 188 614 328	21 146 498 726	**-4,70%**
	cpu	56 :36 :03	56 :20 :16	**-0,46%**

TABLE 2 – $dom/wdeg_{cbj}$ vs $dom/wdeg$

DCSPs		$dom/wdeg_{cbj}$	$dom/wdeg^T_{cbj}$	$dom/wdeg^T_{cbj}$ vs $dom/wdeg_{cbj}$
A_1	#asgs	548 839 044	320 962 468	**-41,52%**
	cpu	00 :57 :51	00 :45 :59	**-20,51%**
B_1	#asgs	1 628 697 906	1 365 322 208	**-16,17%**
	cpu	3 :30 :57	3 :12 :19	**-8,83%**
B_2	#asgs	7 644 525 648	7 398 902 997	**-3,21%**
	cpu	11 :02 :34	10 :22 :58	**-5,98%**
B_3	#asgs	17 250 317 028	18 829 599 249	9,16%
	cpu	26 :05 :21	28 :01 :24	7,41%
B_4	#asgs	18 860 453 983	20 386 953 037	8,09%
	cpu	31 :24 :56	33 :19 :55	6,10%
C_1	#asgs	696 215 827	227 762 909	**-67,29%**
	cpu	1 :58 :34	1 :12 :06	**-39,19%**
C_2	#asgs	2 116 805 967	1 739 005 574	**-17,85%**
	cpu	3 :20 :54	2 :45 :34	**-17,59%**
C_3	#asgs	4 776 536 475	4 802 963 000	0,55%
	cpu	8 :54 :25	8 :58 :31	0,77%
C_4	#asgs	21 146 498 726	5 802 664 153	**-72,56%**
	cpu	56 :20 :16	13 :36 :51	**-75,83%**

TABLE 3 – Mécanisme d'apprentissage