Recherche de la substituabilité par l'arc-cohérence de singleton

Dominique D'Almeida **Lakhdar Saïs**

CRIL-CNRS, Université d'Artois
Rue Jean Souvraz SP18, F-62307 Lens Cedex France

{dalmeida,sais}@cril.fr

Résumé

Dans ce papier, une généralisation sémantique de la substituabilité des valeurs au voisinage est présentée. Notre généralisation est obtenue par une substitution de la notion syntaxique de supports avec une mesure sémantique de la propagation de valeurs obtenues par une utilisation classique de l'arc-cohérence de singleton. Ces nouvelles formes généralisées sont alors exploitées de deux manières différentes. Dans un premier temps, un nouveau prétraitement, intégré à l'algorithme d'Arc-Cohérence de Singleton, des réseaux de contraintes est proposé. Ensuite, puisque l'arc-cohérence de singleton est une opération de base des algorithmes de résolution de type MAC (choix+propagation), nous montrons que notre généralisation peut être facilement exploitée dynamiquement. Les résultats expérimentaux de notre approche se montrent intéressants sur certaines classes d'instances CSP pour le prétraitement, et permettent de montrer que la recherche dynamique de valeurs substituables aux algorithmes de recherche classiques s'avère rentable.

1 Introduction

Le problème de satisfaction de contraintes (CSP) est un cadre général de modélisation et de résolution de problèmes combinatoires, dans notre cas, à domaines finis discrets. Il est défini comme le problème consistant à trouver des valeurs à affecter aux variables de manière à satisfaire des contraintes généralement exprimées par l'ensemble des combinaisons de valeurs autorisées (ou, dualement, interdites). Ce problème de décision NP-Complet, a conduit de nombreux chercheurs à mettre en œuvre des approches algorithmiques pour résoudre aussi efficacement que possible le cas général. Ceci a permis de définir de nouveaux paradigmes de résolution et à en améliorer les étapes, telle que le prétraitement, la propagation de contraintes, le choix de la variable ou le choix de la valeur à affecter. Dans le cas d'une recherche arborescente, il est possible d'entrevoir deux types de stratégies, non exhaustives, qui, a un instant τ de la recherche, visent à comprendre et à analyser ce qui s'est passé avant τ pour mieux anticiper le futur de la recherche.

Pour les stratégies prenant en compte le passé (approches rétrospectives), l'analyse de conflits et la génération de nogoods (ensembles de choix d'affectations incohérents) restent des moyens efficaces d'élaguer un espace de recherche, c'est-à-dire d'éviter d'évaluer un sous-espace de recherche redondant qui mènerait à une incohérence, mais peuvent être coûteuses ou difficiles à mettre en place. Pour les stratégies privilégiant le futur (approches prospectives), il est possible de considérer les techniques de propagations de contraintes, par exemple, qui permettent de supprimer les valeurs qui s'avèrent problèmatiques (incohérentes) dans la suite de la résolution. Mais il y a aussi des traitements efficaces prenant en compte ces deux types de stratégies. L'un des exemples est l'heuristique de choix de variables dom/wdeg qui utilise la pondération de contraintes préalablement falsifiées pour choisir la prochaine variable à affecter.

D'autres types de traitements ont aussi fait l'objet de nombreux travaux, comme la détection et l'exploitation de différentes formes de structures du problème. Parmi elles, nous pouvons citer les symétries de manière générale [1, 6], et des formes plus faibles telles que l'interchangeabilité et la substituabilité de valeurs au voisinage [3]. Ces dernières sont plus simples à détecter (inclusions de tuples autorisés au voisinage) et

à exploiter (suppressions de valeurs) que les symétries dont le problème de la détection, équivalent à l'automorphisme de graphes, est difficile dans le cas général. Ces structures sont détectées, de manière générale, dans leurs formes syntaxiques au niveau du prétraitement et exploitées ensuite de manière statique ou dynamique [2, 4].

Dans ce cadre, l'approche que nous proposons est basée sur une généralisation sémantique de la substituabilité et, parallèlement, de l'interchangeabilité de valeurs au voisinage. Celles-ci sont effectuées en remplaçant la notion syntaxique de supports par une mesure sémantique de la propagation des contraintes obtenues par une utilisation classique de l'arc-cohérence de singleton. Cette généralisation admet plusieurs avantages. Premièrement, elle peut être mise en œuvre de manière statique dans le cadre d'un prétraitement intégré à l'algorithme d'Arc-Cohérence de Singleton (SAC) en donnant lieu à une double généralisation de cet algorithme et de la substituabilité au voisinage. Ce prétraitement vise à faciliter la détection de valeurs substituables sans surcoût pendant la recherche. Passer du temps pour le prétraitement peut avoir des conséquences avantageuses pour la suite de la résolution, et le filtrage par SAC en est un exemple. Le second avantage réside dans la simplicité de mise en œuvre dynamique. En effet, comme la vérification de la cohérence d'arc des singletons est une opération de base des algorithmes de type MAC, il est possible d'exploiter cette généralisation de manière dynamique au moment de l'affectation d'une valeur d'une variable. Celle-ci combine donc les stratégies "passé" et "futur", et permet d'agir en cours de recherche. Elle permet de conserver une trace de l'état du réseau obtenu par la propagation des contraintes après affectation d'une variable, pour vérifier par la suite si cet état réapparaît et éviter ainsi une exploration redondante.

Après avoir défini quelques notions de bases essentielles à la compréhension du problème, nous décrirons notre généralisation de la substituabilité au voisinage par l'arc-cohérence de singleton (3.1), en étudiant les approches statiques (3.2) et dynamiques du problème (3.3). La section 4 présentera la validation expérimentale de ces approches avant de conlure en énonçant quelques perspectives.

2 Notion de bases

Un réseau de contraintes $\mathcal{P}$ est un couple $(\mathcal{X}, \mathcal{C})$ avec $\mathcal{X} = \{x_1, ..., x_n\}$ un ensemble de variables et $\mathcal{C} = \{c_1, ..., c_e\}$ un ensemble de contraintes entre les variables de $\mathcal{X}$. À chaque variable x_i de $\mathcal{X}$ est associé un ensemble de définition fini $dom(x_i) = \{v_1, ..., v_d\}$, appelé domaine de x_i. Une affectation (resp. réfutation) est un couple $(x, v) \in (\mathcal{X}, dom(x))$, noté $x = v$ (resp. $x \neq v$), tel que $dom(x) = \{v\}$ (resp. v est supprimée de $dom(x)$).

Chaque contrainte c_i est définie par un couple (s_i, r_i) tel que :

- $s_i \subseteq \mathcal{X}$ est appelé portée de la contrainte c_i et correspond aux variables sur lesquelles s'applique la contrainte c_i. On note $card(s_i) = a_i$ l'arité de la contrainte c_i ;
- $r_i \subseteq \prod_{x \in s_i} dom(x)$ est appelée relation et représente un ensemble de tuples autorisés, appelés supports de c_i, entre les variables de s_i.

Relativement à la portée, deux variables distinctes x et y sont dites voisines ssi il existe une contrainte $c_i \in \mathcal{C}$, appelée contrainte voisine de x et y, telle que $\{x, y\} \subseteq s_i$. Un ensemble de variables $X \subseteq \mathcal{X}$ (resp. ensemble de contraintes $C \subseteq \mathcal{C}$) est au voisinage d'une variable x ($x \notin X$) si et seulement si pour tout $x_i \in X$ (resp. $c_i \in C$), x est voisine de x_i (resp. c_i).

En considérant la notion de tuple, nous noterons $dom(c_i)$ le domaine de définition de c_i tel que $dom(c_i) = r_i$. Chaque tuple $t \in dom(c_i)$ est défini par un $a_i - uplet$ $(v_1, ..., v_{a_i})$. Nous utiliserons la notation $t[x_j]$ pour désigner la valeur v_j associée à x_j et $t[\hat{x_j}]$ pour représenter $(v_1, ..., v_{j-1}, v_{j+1}, ..., v_{a_i})$. Un tuple $t \in dom(c_i)$ est un support pour $x = v$ ($x \in s_i$ et $v \in dom(x)$) si et seulement si $t[x] = v$. L'ensemble des supports pour $x = v$ dans c_i est noté $supports(c_i|_{x=v})$. Une contrainte est satisfaite ssi elle contient au moins un support.

Une solution pour un réseau de contraintes $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ est une affectation de chaque variable de $\mathcal{X}$ satisfaisant toutes les contraintes. Un réseaux de contraintes est dit cohérent (ou satisfaisable) s'il admet au moins une solution, et incohérent (ou insatisfaisable) sinon. Le problème de satisfaction de contraintes CSP est le problème de décision (NP-complet) qui consiste à déterminer si un réseau de contraintes admet ou non une solution. Nous évoquerons la résolution d'une instance CSP, définie par un réseau de contraintes $\mathcal{P}$, comme étant la recherche d'une solution ou la démonstration de l'insatisfaisabilité.

Pour la résolution de CSP, nous considérons uniquement des algorithmes de recherche en profondeur d'abord, de type retours-arrière (Backtrack ou BT), maintenant la cohérence d'arc à chaque noeud de l'arbre (MAC pour Maintain Arc Consistency). Cet algorithme de type choix+propagation est complet, c'est-à-dire qu'il amène nécessairement à une réponse en parcourant, dans le pire des cas, l'ensemble de l'espace de recherche.

La Cohérence d'Arc est un processus de filtrage appelé propagation de contraintes. Dans un CSP une valeur v de $dom(x)$ est arc-cohérente pour une contrainte

c_i ($x \in s_i$) ssi il existe un support pour $x = v$ dans c_i. Une variable est arc-cohérente pour une contrainte ssi toutes ses valeurs sont arc-cohérentes. Une variable est arc-cohérente ssi elle est arc-cohérente pour toutes les contraintes dans lesquelles cette variable est impliquée. Un réseau est arc-cohérent ssi toutes ses variables sont arc-cohérentes. Sans distinction sur l'arité des contraintes du réseau, la Cohérence d'Arc est appelée Cohérence d'Arc Généralisée (GAC) D'autres types de cohérences existent et permettent un filtrage, plus ou moins fort, des variables en fonction des contraintes du réseau. On définit $\phi(\mathcal{P})$ le réseau de contraintes obtenu après l'application de l'algorithme de propagation de contraintes ϕ. Par exemple, ϕ = GAC signifie que toutes les valeurs arc-incohérentes de $\mathcal{P}$ sont supprimées. Dans la suite nous noterons, pour simplifier, le réseau de contraintes obtenu par l'application de la Cohérence d'Arc AC($\mathcal{P}$). S'il existe une variable ayant un domaine vide dans $\phi(\mathcal{P})$, on note $\phi(\mathcal{P}) \models \bot$ l'incohérence résultant de ϕ. Le sous-réseau obtenu après l'affectation $x = v$ est noté $\mathcal{P}|_{x=v}$.

La résolution de type MAC impose de spécifier le type de branchement utilisé pour développer l'arbre de recherche. En effet, il est possible de considérer un branchement binaire (2way-branching) ou non binaire (dway-branching). Par 2way-branching , on entend qu'à chaque étape un couple variable-valeur (x, v) est sélectionné et deux cas sont ensuite considérés : l'affectation $x = v$ et, si celle-ci n'aboutit pas à une solution, la réfutation $x \neq v$ qui se poursuit du choix d'un nouveau couple. Pour le $d$way-branching , à chaque étape, une variable x est sélectionnée et affectée à une valeur v de $dom(x)$. Si cette valeur est réfutée, une autre valeur de $dom(x)$ est choisie puis affectée, ceci tant que la réfutation n'amène pas à une incohérence locale. Chacun de ces modèles a des avantages, le 2way-branching est néanmoins préféré en pratique puisqu'il donne une importance majeure à l'heuristique de choix de variables.

En suivant ces modèles axés sur le développement d'un arbre de recherche, nous pouvons définir une notion d'instanciation pour un CSP $\mathcal{P} = (\mathcal{X}, \mathcal{C})$. L'instanciation des variables de $X \subseteq \mathcal{X}$ est une application $\mathcal{I}_X$ de X dans $dom(x)$ qui a x associe v, décrivant l'affectation $x = v$. $\mathcal{I}_X$ est appelée instanciation complète si $X = \mathcal{X}$ et instanciation partielle sinon. $\mathcal{I}_\emptyset$ représentera l'absence d'affectation. Une instanciation $\mathcal{I}_X$ est cohérente si et seulement si elle vérifie les contraintes du réseau. Une instanciation complète et cohérente est une solution.

Ces quelques notions permettent de définir notre cadre, dont le travail préliminaire passe par la définition et l'étude de la substituabilité au voisinage, ce qui permettra une meilleure compréhension de l'intérêt d'intégration de cette approche aux algorithmes existant.

3 Substituabilité basée sur l'arc-cohérence de singleton

3.1 Définition et étude de la substituabilité au voisinage

La substituabilité et, parallèlement, l'interchangeabilité sont les maîtres-mots de notre contribution. Ces propriétés des CSP ont donné lieu à de nombreux travaux mettant en évidence la difficulté à les déterminer [3]. Il existe deux formes de substituabilité et interchangeabilité : complète ou local.

Pour un CSP $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ et deux couples distincts $(x, v), (x, v') \in (\mathcal{X}, dom(x))$, v est complètement substituable par v' ssi pour chaque solution $\mathcal{I}_\mathcal{X}$ telle que $\mathcal{I}_\mathcal{X}(x) = v$, il existe une solution $\mathcal{I}'_\mathcal{X}$ telle que $\mathcal{I}'_\mathcal{X}(x) = v'$ et $\mathcal{I}'_\mathcal{X}(y) = \mathcal{I}_\mathcal{X}(y)$ pour tout $y \in \mathcal{X}$ et $y \neq x$. Il est facile de voir que les notions d'interchangeabilité complète et de substituabilité complète sont liées. En effet, deux valeurs v et v' sont complètement interchangeables ssi v est complètement substituable par v' et v' est complètement substituable par v. Deux valeurs complètement interchangeables sont donc complètement substituables. Cette définition montre la difficulté à déterminer l'interchangeabilité ou la substituabilité complète de deux valeurs, dont la complexité est coNP-complet. Cela nécessite l'énumération des solutions (variable de référence exclue) et la vérification de l'inclusion de chacune dans les autres.

Cependant, un affaiblissement permet de définir une forme locale de la substituabilité et de l'interchangeabilité de deux valeurs : substituabilité au voisinage et interchangeabilité au voisinage. Celles-ci se définissent par la substituabilité ou l'interchangeabilité entre deux valeurs d'une variable x sur un sous-ensemble de variables ou de contraintes du réseau, notamment au voisinage de x. Dans la suite, nous nous concentrerons sur la substituabilité au voisinage, que nous privilégierons à l'étude de l'interchangeabilité car, comme nous l'évoquerons, déterminer la substituabilité suffit dans notre cadre. De façon analogue à la forme de substituabilité complète, il nous est possible de définir la substituabilité au voisinage comme suit :

Définition 1 *Soit $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ un CSP, soient deux couples distincts $(x, v), (x, v') \in (\mathcal{X}, dom(x))$ et X l'ensemble de variables composé de x et de ses voisines. v est substituable par v' au voisinage de x si et seulement si, pour chaque instanciation partielle cohérente $\mathcal{I}_X$ telle que $\mathcal{I}_X(x) = v$, il existe une instanciation partielle cohérente $\mathcal{I}'_X$ telle que $\mathcal{I}'_X(x) = v'$ et $\mathcal{I}'_X(y) = \mathcal{I}_X(y)$, pour tout $y \in X$ et $y \neq x$.*

Cette définition se concentre cette fois sur l'ensemble de variables X, correspondant à la variable x et son voisinage, pour mettre en évidence la relation entre l'état du voisinage de x pour l'une et l'autre des valeurs de son domaine. Dans la littérature, il est possible de trouver une reformulation plus classique de cette définition [3] qui se concentre sur la notion de supports au voisinage :

Définition 2 *Pour un* CSP $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ *et deux couples* $(x, v), (x, v') \in (\mathcal{X}, dom(x))$. *$v$ est substituable par v' au voisinage de x ssi, pour toute contrainte c_i voisine de x, $supports(c_i|_{x=v}) \subseteq supports(c_i|_{x=v'})$.*

Intuitivement, la notion de supports peut être associée à la notion de propagation de contraintes pour énoncer les propositions suivantes :

Proposition 1 *Soit $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ un* CSP*, soient deux couples $(x, v), (x, v') \in (\mathcal{X}, dom(x))$, et ϕ un algorithme de propagation de contraintes. v est substituable par v' au voisinage de x si et seulement si le* CSP *$\phi(\mathcal{P}|_{x=v})$ est un sous-réseau de $\phi(\mathcal{P}|_{x=v'})$, à l'exception de la variable x.*

Proposition 2 *Soit $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ un* CSP*, soient deux couples $(x, v), (x, v') \in (\mathcal{X}, dom(x))$ tels que v est substituable par v' au voisinage de x , et ϕ un algorithme de propagation de contraintes. Si $\phi(\mathcal{P}|_{x=v'}) \models \bot$ alors $\phi(\mathcal{P}|_{x=v}) \models \bot$.*

Ces deux résultats forment le centre d'intérêt de notre contribution. Pour prouver ces propositions, il suffit de se concentrer sur les définitions 1 et 2. Le sens "si" est immédiat. Puisque v est substituable par v' au voisinage de x, nous avons, d'après la définition 2, $supports(c_i|_{x=v}) \subseteq supports(c_i|_{x=v'})$. En itérant la propagation des contraintes jusqu'à atteindre un point fixe du CSP, les supports $supports(c|_{x=v})$ de $\phi(\mathcal{P}|_{x=v})$ sont inclus dans les supports $supports(c|_{x=v'})$ pour tout c, à l'exception de la variable x . Donc CSP $\phi(\mathcal{P}|_{x=v})$ est un sous-réseau de $\phi(\mathcal{P}|_{x=v'})$, variable x exclue.

Dans l'autre sens, nous avons $\phi(\mathcal{P}|_{x=v})$ sous-réseau de $\phi(\mathcal{P}|_{x=v'})$, en omettant la variable x, et ϕ un algorithme vérifiant la cohérence locale du réseau. Donc pour chaque instanciation localement cohérente $\mathcal{I}_{X\setminus\{x\}}$ (avec X ensemble de variable composé de x et de son voisinage) de $\phi(\mathcal{P}|_{x=v})$, il existe une instanciation localement cohérente $\mathcal{I}'_{X\setminus\{x\}}$ telle que $\mathcal{I}'_{X\setminus\{x\}}(y) = \mathcal{I}_{X\setminus\{x\}}(y)$, pour tout $y \in X$ et $y \neq x$. En considérant la variable x dans chacune des instanciations, nous obtenons $\mathcal{I}_X(x) = v$ et $\mathcal{I}'_X(x) = v'$, mais aussi $\mathcal{I}'_{X\setminus\{x\}}(y) = \mathcal{I}_{X\setminus\{x\}}(y)$. D'après la définition 1, v est donc substituable par v' au voisinage de x. Enfin, trivialement, en considérant la proposition 1, chaque solution, variable x exclue, du CSP $\phi(\mathcal{P}|_{x=v})$ l'est aussi pour $\phi(\mathcal{P}|_{x=v'})$. Par contraposée, si le CSP $\phi(\mathcal{P}|_{x=v'})$ est incohérent, alors $\phi(\mathcal{P}|_{x=v})$ l'est aussi. Ce qui montre la proposition 2.

L'idée est donc d'utiliser et d'intégrer le plus efficacement possible ces résultats aux algorithmes de prétraitement et de recherche comme, par exemple, SAC et MAC. Avant d'exploiter les valeurs substituables au voisinage, il faut néanmoins remplir cette condition de substituabilité. Pour cela, il suffit de se concentrer sur la définition 2 et de vérifier l'inclusion des supports des contraintes voisines pour les affectations $x = v$ et $x = v'$. Notons que l'utilisation de tuples autorisés, pour le calcul de l'inclusion, est privilégiée dans nos descriptions mais est, dans le cas général, équivalente pour la vérification d'inclusion à l'utilisation des tuples interdits. Aussi, une remarque peut être faite quant à l'efficacité, en nombre de valeurs substituables, de la détection de la substituabilité au voisinage, puisque celle-ci est dépendante de l'algorithme de propagation de contraintes. Dans la suite, nous concentrerons notre étude sur la propagation par Cohérence d'Arc, notamment utilisé dans les algorithmes SAC et BT+MAC.

3.2 Substituabilité au prétraitement SNS

Pour correspondre au mieux à notre étude de la substituabilité au voisinage, le choix de l'intégration au prétraitement se porte sur l'algorithme SAC. Pour un CSP $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ et une variable $x \in \mathcal{X}$, une valeur $v \in dom(x)$ est singleton arc-cohérente (sac) ssi $AC(\mathcal{P}|_{x=v}) \not\models \bot$. Et, de manière étendue, une variable x est sac ssi, pour tout $v \in dom(x)$, v est sac. L'Arc-Cohérence de Singleton (SAC) est une forme plus forte de cohérence locale que GAC qui garantit $\phi(\mathcal{P}|_{x=v}) \not\models \bot$, pour chaque couple variable-valeur (x, v). La propagation suivant l'affectation de chaque couple (x, v) présente un grand intérêt dans notre cas puisque nous pourrons profiter du traitement efficace effectué par cet algorithme et y greffer notre approche, tout en bénéficiant du filtrage effectué par SAC.

L'algorithme résultant de l'intégration de notre approche, que nous nommons SNS pour "*SAC and Neighbouhood Substituability*", se définit en deux étapes. La première se concentre sur la détection des valeurs substituables. Pour cela, les supports des contraintes voisines sont stockés après l'appel à l'algorithme AC sur le réseau $\mathcal{P}|_{x=v}$, dans une structure correspondant à un état du voisinage de x (noté $\mathcal{E}_{x=v}$). L'inclusion des états s'effectue ensuite simplement en vérifiant l'inclusion des supports de chaque contrainte de l'un et l'autre des états. Formellement, en considérant $\mathcal{E}_{x=v}(c)$ la contrainte c dans $\mathcal{E}_{x=v}$, nous avons $\mathcal{E}_{x=v} \subseteq \mathcal{E}_{x=v'}$ ssi, pour tout tuple $t \in \mathcal{E}_{x=v}(c)$, il existe un tuple

$t' \in \mathcal{E}_{x=v'}$ tel que $t[\hat{x}] = t'[\hat{x}]$. S'il y a inclusion pour toutes les contraintes, il y a inclusion pour l'état. La seconde étape utilise la proposition 1. Dans le cadre de SNS, si v est substituable par v' au voisinage de x, les solutions obtenues avec l'affectation $x = v$ permettent de déduire certaines solutions pour $x = v'$, donc v est supprimée. La détection de l'interchangeabilité au voisinage n'apporte donc pas de nouvelles informations puisque l'une ou l'autre des variables est destinée à être supprimée.

Associée à SAC, cette approche, présentée dans l'algorithme 1, permet donc un filtrage des valeurs "non sac" et substituables au voisinage, ce qui le rend plus fort que l'algorithme SAC. Dans celui-ci, seules les instructions 7 à 11 se concentrent sur la recherche de valeurs substituables, le reste de l'algorithme correspond à SAC, ce qui montre la facilité d'intégration.

Algorithme 1 : SNS

```
   Entrées : P = (X, C) : un CSP
   Sorties : SNS(P)
1  début
2      répéter
3          pour chaque couple (x, v) ∈ (X, dom(x)) faire
4              si AC(P|x=v) ⊨⊥ alors
5                  Supprimer (x, v)
6              sinon
7                  Stocker E_{x=v}
8                  pour chaque (x, v') prouvé sac faire
9                      E_{x=v} ⊆ E_{x=v'} ⇒ Supprimer (x, v)
10                     Et si v n'est pas supprimé :
11                     E_{x=v'} ⊆ E_{x=v} ⇒ Supprimer (x, v')
12                 fin
13             fin
14         fin
15     jusqu'à ne plus avoir de suppression de couples
16 fin
```

Aussi, l'efficacité en temps et en espace d'un tel calcul d'inclusion des supports dépend directement de l'arité de la contrainte. Si la contrainte c implique a variables ayant chacune un domaine de taille d, la contrainte peut avoir, dans le pire des cas, d^a supports.

Avant d'entamer le calcul de la complexité algorithmique de SNS, nous rappelons que le réseau de contraintes possède n variables de domaine d, que le nombre de contraintes est e et chacune est d'arité maximale r.

Pour la complexité en espace, considérons tout d'abord l'espace nécessité par un état pour un couple variable-valeur (x, v). Puisque l'arité des contraintes est r, chaque variable est voisine de $r-1$ variables, dont le domaine est d. Il y a d^{r-1} supports par contrainte et donc $(r-1)d^{r-1}$ valeurs par contrainte. Dans le pire des cas, chaque variable est impliquée dans les e contraintes du réseau, ce qui permet de déduire qu'il y a $e(r-1)d^{r-1}$ valeurs par état, dans le pire des cas. Pour comparer les états correspondant à chaque valeur d'une variable, SNS nécessite de conserver d états. Nous avons donc une complexité en espace en $O(erd^r)$ pour l'ensemble des états à stocker lors de SNS.

Pour calculer la complexité en temps de SNS, il faut tout d'abord considérer que celui-ci est intégré à l'algorithme SAC, puis décomposer SNS. Tout d'abord, il est possible de noter qu'un traitement est effectué pour chaque couple variable-valeur, soit nd fois. Ce traitement correspond à la cohérence d'arc (1), le stockage de l'état associé au couple (2) et la comparaison avec les états existant pour une même variable (3). La complexité de l'algorithme AC est en $O(erd^r)$ (1). Le stockage d'un état nécessite le parcours de ses $e(r-1)d^{r-1}$ valeurs (2). Chacun de ces états doit être comparé aux états précédemment conservés. L'inclusion sera testée dans un sens puis dans l'autre, ce qui nécessite $d-1$ vérifications d'inclusion pour chaque couple variable-valeur. L'inclusion de deux états implique le parcours de l'ensemble des valeurs de ces états, donc erd^r valeurs pour les vérifications d'inclusion (3). En détail, nous avons donc une complexité en temps pour le traitement énoncé précédemment proche de $nd \times (erd^r + e(r-1)d^{r-1} + erd^r)$, soit $O(ernd^{r+1})$. Or, comme dans le cadre de l'algorithme SAC classique, la suppression d'une valeur dans le réseau de contraintes, quelle que soit la raison, implique la réitération du traitement effectué ci-dessus puisque l'état du réseau à changer. Comme le réseau possède nd valeurs, le traitement doit, dans le pire des cas, être itéré nd fois. La complexité en temps, permettant de détecter l'ensemble des valeurs non-sac et les valeurs substituables au voisinage est $O(ern^2d^{r+2})$.

Bien qu'ayant des complexités en espace et en temps élevées à cause de l'arité des contraintes, il faut néanmoins noter qu'en pratique, l'efficacité de l'approche dépendra, bien évidemment, des traitements et des structures utilisées. Aussi, puisque cette complexité est fonction de l'arité, il est intéressant de voir le cas particulier des contraintes binaires, qui peut être considéré comme le cas le plus "léger" pour SNS. La complexité en espace est en $O(end^2)$ et la complexité en temps est $O(en^2d^4)$, c'est-à-dire comparable à l'algorithme SAC classique dans le pire des cas.

La détection de valeurs substituables au voisinage dans le cadre d'un prétraitement peut donc s'avérer intéressant notamment par son intégration facile dans des algorithmes tel que SAC, permettant d'approfondir leurs capacités. Pourtant, les pleines ressources de notre prétraitement ne sont pas développées ici mais sont nombreuses. Il est de notre avis qu'il est possible de déduire des informations sur le réseau, par exemple structurelles, par notre prétraitement pour faciliter l'utilisation d'une approche dynamique.

3.3 Recherche dynamique de la substituabilité

Le développement des différentes étapes de l'approche statique, effectué en 3.2, illustre tous les traitements que nous allons étendre au cas du 2way-branching, permettant de déduire la substituabilité au voisinage dynamiquement. Tout d'abord, l'aspect dynamique de cette intégration vient de la modification constante du réseau de contraintes. Dans le cas d'un algorithme de type BT+MAC, le maintien de l'arc-cohérence dans le réseau se fait quelle que soit la décision qui a été prise, notamment après affectation d'une variable. La détection de valeurs substituables au voisinage d'une variable pourra donc se greffer facilement, comme dans le cadre statique, et être calculée à chaque nœud de l'arbre de recherche.

De manière analogue à l'approche SNS, les supports au voisinage sont stockés après chaque propagation des contraintes suivant une affectation $x = v$. Les états d'une variable sont donc décrits par l'état produit lors de la dernière affectation (l'état courant) et par les états conservés pour les valeurs précédemment réfutées (les états passés). D'après la proposition 2, si l'état courant $\mathcal{E}_{x=v}$ est inclus dans un état passé $\mathcal{E}_{x=v'}$, alors la valeur v est substituable par v' et puisque v' a été réfutée, v doit l'être elle aussi. Notons que ce résultat est dû à la réfutation du couple (x, v') à la racine de l'arbre de recherche, ce qui induit une incohérence globale de celui-ci. Le choix $x = v$, menant à un sous-état de celui produit par $x = v'$, produit nécessairement une incohérence globale de (x, v). L'approche dynamique permet de ne pas à avoir à réévaluer certains sous-espaces de recherche localement cohérents mais globalement incohérents pouvant être rencontrés s'il n'y a pas détection de la substituabilité des valeurs au voisinage d'une variable. Pour cette approche, la détection de l'interchangeabilité au voisinage est donc impossible puisque la vérification d'inclusion est unilatérale et imposée par la recherche. Les lignes 5 à 7 de l'algorithme ns+dual (Algo. 2) illustrent la description effectuée ci-dessus, intégrée à l'algorithme classique 2way-branching de type BT+MAC.

L'intégration de la détection de la substituabilité au 2way-branching peut, bien entendu, être adaptée au cas du *d*way-branching (que nous appelerons ns+dway) de manière similaire puisque les différences entre ces algorithmes sont indépendantes de notre approche. Cependant, malgré la facilité de mise en œuvre, la condition de la ligne 13 de l'algorithme ns+dual souligne un fait important dans notre cadre. La comparaison entre les états doit être faite sous certaines conditions, notamment liées à l'état du réseau avant l'affectation. Par exemple, comparer un état au voisinage de x pour le réseau $\phi(\mathcal{P}|_{y=0,x=0})$ à un état au voisinage de x pour le réseau $\phi(\mathcal{P}|_{y=1,x=1})$ n'a pas de sens dans le cas de la détection de la substituabilité au voisinage, à moins que les réseaux $\phi(\mathcal{P}|_{y=0})$ et $\phi(\mathcal{P}|_{y=1})$ soient équivalents.

Algorithme 2 : ns+dual

```
   Entrées : P = (X, C) : un CSP
   Sorties : ⊤ si P est cohérent, ⊥ sinon
1  début
2      tant que il existe une variable non affectée faire
3          Choisir (x, v) ∈ (X, dom(x)) tel que x non affectée
4          Affecter v à x et propager les contraintes
5          Stocker l'état du voisinage E_{x=v}
6          pour chaque état E_{x=v'} conservé faire
7              E_{x=v} ⊆ E_{x=v'} ⇒ P ⊨ ⊥
8          fin
9          tant que P ⊨ ⊥ et ∃ une variable affectée faire
10             Restaurer le niveau précédent
11             Réfuter le dernier couple (x', v') et Propager
12             si P ⊨ ⊥ alors
13                 Supprimer les états inutiles
14             fin
15         fin
16         si P ⊨ ⊥ et ∄ une variable affectée alors
17             retourner ⊥
18         fin
19     fin
20     retourner ⊤
21 fin
```

Sans nous concentrer sur les cas particuliers, un critère nécessaire à la comparaison de deux états relève des instanciations desquelles ils sont produits. Par exemple, un état obtenu à la racine de l'arbre de recherche peut-il être comparé à des états produits après un nombre quelconque d'affectations ? La réponse s'appuie sur l'utilisation de la proposition 1 :

Proposition 3 *Soit $\mathcal{P} = (\mathcal{X}, \mathcal{C})$ un* CSP*, soient deux couples $(x, v), (x, v') \in (\mathcal{X}, dom(x))$, et ϕ un algorithme de propagation de contraintes. Soit $\mathcal{I}_X$ une instanciation cohérente telle que $X \subseteq \mathcal{X} \setminus \{x\}$. v est substituable par v' au voisinage de x si et seulement si le* CSP *$\phi(\mathcal{P}|_{\mathcal{I}_X, x=v})$ est un sous-réseau de $\phi(\mathcal{P}|_{x=v'})$, à l'exception de la variable x.*

En utilisant la proposition 1, la preuve est immédiate. Si v est substituable par v' au voisinage de x, alors $\phi(\mathcal{P}|_{x=v})$ est un sous-réseau de $\phi(\mathcal{P}|_{x=v'})$, variable x exclue. Quelle que soit l'instanciation $\mathcal{I}_X$ avec $X \subseteq \mathcal{X} \setminus \{x\}$, $\phi(\mathcal{P}|_{\mathcal{I}_X, x=v})$ est un sous-réseau de $\phi(\mathcal{P}|_{x=v})$. Donc $\phi(\mathcal{P}|_{\mathcal{I}_X, x=v})$ est un sous-réseau de $\phi(\mathcal{P}|_{x=v'})$, en exluant la variable x. Dans l'autre sens, si $\phi(\mathcal{P}|_{\mathcal{I}_X, x=v})$ est un sous-réseau de $\phi(\mathcal{P}|_{x=v'})$ alors les supports de $\phi(\mathcal{P}|_{\mathcal{I}_X, x=v})$ sont inclus dans les supports de $\phi(\mathcal{P}|_{x=v'})$, en particulier au voisinage de x. Donc d'après la définition 2, v est substituable par v' au voisinage de x. Ce qui prouve la proposition 3.

Notons néanmoins que ce résultat tient uniquement si la vérification de l'inclusion se fait d'un état issu

d'une instanciation dans un état issu de la racine de l'arbre de recherche. En effet, la réfutation du couple (x, v) à la racine de l'arbre induit une incohérence globale de celui-ci, donc les valeurs sustituables par v au voisinage de x sont donc incohérentes localement, relativement à l'état du voisinage, qui est le résultat d'une l'instanciation. Or, dans l'autre sens, l'incohérence résultant d'une instanciation ne permet pas de déduire, dans le cas général, une incohérence globale, et donc ne permet pas de déduire, toujours dans le cas général, la suppression de valeurs affectées à la racine de l'arbre.

De plus, ce résultat peut être élargi aux réseaux $\phi(\mathcal{P}|_{\mathcal{I}_X, x=v})$ et $\phi(\mathcal{P}|_{\mathcal{I}_Y, x=v'})$, pour deux instanciations $\mathcal{I}_X$ et $\mathcal{I}_Y$ avec $Y \subseteq X$. Si l'on considère $\mathcal{P}' = \phi(\mathcal{P}|_{\mathcal{I}_Y})$, nous retrouvons la proposition 3 pour les réseaux $\phi(\mathcal{P}'|_{\mathcal{I}_{X\setminus Y}, x=v})$ et $\phi(\mathcal{P}'|_{x=v'})$. La remarque précédente visant le sens d'inclusion démontre donc pourquoi la condition $Y \subset X$ est imposée pour l'instanciation.

La comparaison entre les états du voisinage pour une même variable est donc réduite, dans notre cadre d'étude, à la comparaison de l'état courant issu d'une instanciation et des états passés issus d'une restriction de cette instanciation, exception faite de la variable pour laquelle les valeurs substituables sont recherchées. Ce critère de comparaison est d'une grande utilité dans le cas du 2way-branching, puisqu'il permet une détection de valeurs substituables au voisinage malgré les affectations effectuées à différents niveaux de la recherche. Ceci s'applique aussi dans le cadre du $d$way-branching, mais la particularité dûe au choix de la variable à affecter permettra aux états d'être conservés pour une même instanciation, jusqu'à la réfutation complète du domaine de la variable.

Avant d'étudier la complexité algorithmique de l'approche dynamique, nous rappelons que nous étudions un réseau de contraintes ayant n variables, de domaines de taille d, chacune impliquée dans e d'arité maximale r. Dans le cas présent, l'étude de la complexité pour l'adaptation ns+dual ou ns+dway est équivalente, nous ne faisons donc pas d'étude spécifique à chacune de ces approches.

Pour la complexité en espace, le pire des cas correspond à la réfutation de d valeurs par variable, qui ont donc nécéssité le stockage de d états. Les n variables imposent donc de conserver nd états pour l'ensemble de l'évaluation dans l'arbre de recherche, dans le pire des cas. Puisque chaque état est composé de $e(r-1)d^{r-1}$ valeurs, la complexité en espace dans le pire des cas est $O(ernd^r)$, correspondant au nombre de valeurs à conserver pendant la recherche.

Calculer la complexité en temps dans le pire des cas dépend de la façon de traduire le pire des cas issu du développement d'un arbre de taille exponentiel. Notre choix se porte sur l'étude de la complexité en temps correspondant à la détection de la substituabilité des valeurs d'une variable avant que celle-ci ne soit réfutée. Cette réfutation a lieu lorsque d valeurs ont été réfutées et chaque i^{me} affectation conduit à vérifier les états des $i-1$ valeurs réfutées précédemment. Il y a donc, dans le pire des cas, $\binom{d}{2}$ comparaisons d'états. Puisque chaque comparaison nécéssite le parcours de $e(r-1)d^{r-1}$ valeurs, la complexité en temps dans le pire des case est $O(erd^{r+1})$ pour comparer l'ensemble des états avant réfutation complète de la variable.

Pour fournir un ordre d'idée sur le cas binaire, la complexité en espace dans le pire cas est $O(end^2)$ et la complexité en temps dans le pire des cas est $O(ed^3)$ pour déterminer s'il existe des valeurs substituables avant réfutation complète de la variable.

Après l'étude théorique de notre approche, nous allons déterminer si celle-ci s'avère intéressante en pratique.

4 Expérimentations

Avant de développer les différents cadres étudiés précédemment, quelques petites précisions sur le protocole expérimental. Celui-ci se déroule en 2 phases : approche statique intégrée à SAC avec méthode de recherche 2way-branching classique (appelée SNS), et approche dynamique intégrée à ns+dual avec prétraitement par cohérence d'arc (appelée ns+dual). Chaque approche a été comparée à son homologue dit "de base", c'est-à-dire sans détection de substituabilité au voisinage (respectivement SAC et dual). Les expérimentations ont été réalisées dans le cadre binaire essentiellement. Le cas des instances n-aires fait toujours l'objet d'étude pour la mise en pratique, et les complexités en espace et en temps expliquent ce choix. Certaines approches auraient pu être exploitées, comme imposer une limite à l'arité des contraintes évaluées, mais puisque ceci aurait été complètement arbitraire par manque d'études, nous avons décidé de ne présenter que le cas binaire. L'implémentation a été réalisée en JAVA à l'aide du solveur Abscon109 [5] développé au CRIL, et les expérimentations ont été menées sur un processeur Xéon 3GHz avec 1Go de RAM. Les instances CSP utilisées sont celles de la 3^{eme} compétition internationale de solveurs CSP réalisée en 2008 (http ://cpai.ucc.ie/08/). Ces instances sont normalisées, c'est-à-dire que deux contraintes ne peuvent pas impliquer le même ensemble de variables. Les instances sont divisées en 4 catégories contraintes en extension/intension et SAT/UNSAT. Le temps de résolution a été limité à $1200s$ et la mémoire à 900Mo.

4.1 Intégration statique

La comparaison dans le cadre statique est basée sur les critères suivants : le nombre d'instances résolues, le nombre de valeurs filtrées, les temps de résolution/prétraitement et le nombre de nœuds parcourus.

Extension SAT

Solveur	#	Temps	# inst.	Filtr.
SAC	296	20624	57	4373
SNS	296	21019	72	41909

Extension UNSAT

Solveur	#	Temps	# inst.	Filtr.
SAC	195	13824	68	6869
SNS	195	13807	78	8372

Intension SAT

Solveur	#	Temps	# inst.	Filtr.
SAC	253	14867	57	16404
SNS	254	18054	110	82432
−SNS	253	17855	109	79618

Intension UNSAT

Solveur	#	Temps	# inst.	Filtr.
SAC	202	8307	47	24382
SNS	205	10857	51	41546
−SNS	202	8017	49	37690

TABLE 1 – Résultats globaux : SAC contre SNS

La tableau 1 présente les résultats généraux de la résolution de toutes les instances binaires. Dans celle-ci, chaque tableau représente une catégorie, pour lesquelles le nombre d'instances (#), le temps de résolution total en seconde (Temps), le nombre d'instances sur lesquelles au moins une valeur a été supprimée (# inst.) et le nombre de valeurs filtrées (Filtr.). Nous considérons le nombre de valeurs filtrées comme étant le nombre de valeurs supprimées au prétraitement, c'est-à-dire les valeurs non-sac et les valeurs substituables dans le cas de SNS. Aussi, le nom d'une approche précédée d'un signe − indique que seules les instances résolues par les deux approches sont prises en compte dans les résultats de celle-ci.

Premier résultat : l'approche basée sur SNS est meilleure en nombre d'instances résolues que l'approche SAC. Elle répond aux problèmes pour 4 instances de plus que l'approche classique. Or SNS n'a joué un rôle que pour 3 d'entre elles. Nous pouvons donc déduire que les caractéristiques spécifiques de l'approche SNS ont permis de conclure pour 3 problèmes de plus que SAC. Un résultat tout aussi intéressant, toutes catégories confondues, est que le nombre de valeurs filtrées par SAC est de 59771, alors que SNS filtre 182041 valeurs, ce qui montrent la puissance de filtrage de notre approche. Cependant, en temps de résolution total, notre approche n'a pas vraiment l'avantage, puisqu'elle est en dessous de l'approche classique. Elle n'a l'avantage en temps que pour la catégorie "intensions/unsat", dont le temps résolution des instances est le plus faible.

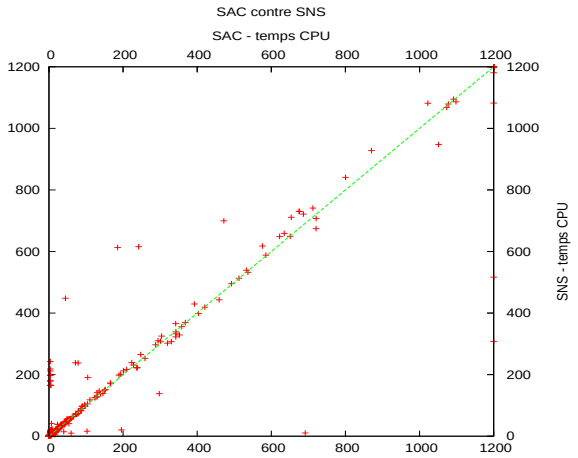

FIGURE 1 – Comparaison des temps CPU : SAC contre SNS

Le nuage de points Figure 1 permet d'avoir une vue globale des temps de résolution des instances. Sur celui-ci chacun des points représente un problème, dont les coordonnées sont désignées par les temps de résolution des instances. Par exemple, la droite $x = 1200$ montre les 4 instances résolues par SNS et non par SAC. Il est facile de déduire de cette figure que SNS est plus mauvais en temps, notamment en observant une nuée de point proche de $(0, 200)$. Malgré ces quelques instances, la majorité des points se situent autour de la droite $y = x$, ce qui montre des temps très proches pour les deux approches.

Même si l'approche s'avère mauvaise pour la recherche de performance, certaines classes d'instances ressortent de ces résultats, notamment celles formant le groupement de point énoncé précédemment. Une analyse plus détaillée permet d'extraire deux classes de problème : JobShop et Taillard OpenShop.

Tout d'abord, les problèmes jobShop utilisés, étudiés par Sadeh-Fox, sont des problèmes *sat* de 50 variables, de domaines de taille comprise entre 100 à 250, et de 265 contraintes binaires. La plupart de ces instances ont été résolues facilement par l'approche SAC, sans pour autant bénéficiée de filtrage. Sur 46 instances résolues par les deux approches, 29 sont triviales, c'est-à-dire que le nombre d'affectations équivaut au nombre de variables.

Solver	#	Trival	Temps	Filtr.	Prepro.	Nœuds
SAC	45	29	175	0	85	208181
SNS	45	35	2369	19293	2208	8801

TABLE 2 – Synthèses des résultats des problèmes jobShop

Le tableau 2 présente une comparaison des deux approches étudiées, dont les critères sont le nombre d'ins-

tances résolus (#), le nombre d'instances trivialement satisfaites (Trivial), le temps de résolution en secondes (Temps), le nombre de valeurs filtrées (Filtr.), le temps de prétraitement en secondes (Prepro.) et le nombre de nœuds de l'arbre de recherche (Nœuds). Ceci montre l'efficacité du prétraitement effectué par SNS pour le critère de la suppression de valeurs. Quelques instances sont trivialement satisfaites et le nombre de nœuds liés à la recherche est fortement réduit. Cependant, le temps de prétraitement occupe la majeure partie du temps de résolution (93%), et c'est sur ce critère que notre approche démontre des lacunes. SNS permet de résoudre en $10s$ (dont $8s$ de prétraitement) une $46e$ instance que SAC résoud en $691s$ (dont $2s$ de prétraitement). Notre approche filtre 517 valeurs et réduisant ainsi le nombre de nœuds parcourus à 192, alors que l'approche classique ne permet aucun filtrage, et a parcouru plus de 4 millions de nœuds. Intégré ce résultat à notre synthèse n'aurait donc pas été pertinent.

Pour terminer l'analyse de l'approche statique, étudions la deuxième classe d'instances intéressante : les instances Taillard OpenShop. Ces problèmes d'optimisations sont des instances dont le nombre de variables est n^2, de taille maximale avoisinant les 300 valeurs et le nombre de contraintes est de $4n^2$ pour $n \in \{4, 5, 7, 10, 20\}$ dans notre cas. 43 instances ont été résolues par les deux approches, et celles-ci n'effectuent aucun filtrage pour 18 d'entre elles. Notons que le temps de prétraitement pour ces 18 instances est d'environ $305s$ pour SNS et $303s$ pour SAC, ce qui est intéressant puisque la vérification des états n'a pas été trop coûteuse en temps.

Solver	#	Trivial	Temps	Filtr.	Prepro	Nœuds
SAC	25	3	1161	10850	590	870351
SNS	25	4	1950	67105	1775	151663

TABLE 3 – Synthèses des résultats des problèmes Taillard-OS

Le tableau 3 se composent des mêmes critères que pour les jobShop. Nous pouvons y effectuer les mêmes remarques que précédemment. Le manque d'efficacité en temps vient du prétraitement qui fait un travail de filtrage très important (90% du temps de résolution), pour évaluer un nombre de nœuds bien plus faible.

Chacune des catégories fait ressortir une classe de problèmes particulière, plus ou moins facile à résoudre. Pour autant, il est difficile de déterminer une classe d'instances complète pour laquelle l'approche SNS est meilleure en temps. Notre prétraitement s'avère très efficace pour la détection jointe de valeurs substituables et non-sac, mais ceci a un coût. Bien que le prétraitement soit peu efficace en temps lorsque ce nombre de valeurs est important, il facilite largement la recherche en réduisant fortement le nombre de nœuds de l'arbre, ce qui lui permet de résoudre plus d'instances que son homologue de base.

4.2 Intégration dynamique

Pour notre approche dynamique, les critères de comparaisons se concentrent sur le nombre d'instances résolues, le temps de résolution en seconde et le nombre de filtrages effectués durant la recherche.

Extension SAT

Solveur	#	Temps	# inst.	Filtr.
dual	296	20845	-	-
ns+dual	294	17972	28	129
−dual	294	18600	-	-

Extension UNSAT

Solveur	#	Temps	# inst.	Filtr.
dual	196	13771	-	-
SNS	196	13716	36	66302

Intension SAT

Solveur	#	Temps	# inst.	Filtr.
dual	253	9391	-	-
ns+dual	254	9465	58	22439
−dual	251	9466	-	-
−ns+dual	251	8706	55	18441

Intension UNSAT

Solveur	#	Temps	# inst.	Filtr.
dual	202	8268	-	-
ns+dual	203	8435	29	152014
−ns+dual	202	7789	28	104917

TABLE 4 – Résultats globaux : dual contre ns+dual

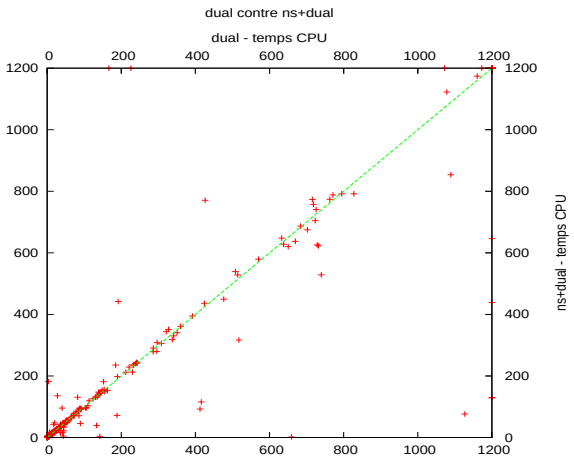

FIGURE 2 – Comparaison des temps CPU : dual contre ns+dual

Le tableau 4 montre que le nombre d'instances résolues est le même pour l'une et l'autre des approches. Le cumule des temps ne fournit pas non plus un critère intéressant pour l'approche dynamique, puisque l'on ne note qu'un très léger gain. Néanmoins, pour les instances communes résolues, l'approche ns+dual

est toujours meilleure, ce que l'on remarque aussi sur le nuage de points 2. Un résultat décevant toutefois : 10 à 20% des instances bénéficient de la détection de valeurs substituables.

Malgré le peu d'instances touchées par notre traitement dynamique, il est possible de dégager quelques classes de problèmes, notamment les deux classes étudiées pour l'approche statique. Les jobShop, tout d'abord, sur 46 problèmes résolus par les deux approches, 29 sont trivialement *sat*. Sur les 17 instances restantes, 10 sont touchées par la détection de la substituabilité, dont l'instance que nous avions écarté de notre synthèse. L'approche dual met près de $660s$ pour répondre à la satisfaisabilité, alors que ns+dual, 1 petite seconde, grâce à la détection de 3 valeurs substituables. Le gain obtenu pour les autres instances restent négligeable puisque celles-ci sont faciles ($15s$ pour ns+dual et $17s$ pour dual).

La deuxième classe correspond aux instances Taillard. Sur 45 instances, 25 sont touchées par la détection de la substituabilité de ns+dual dont 2 instances sont résolues uniquement par ns+dual en filtrant 397 valeurs. dual met $1724s$ tandis que ns+dual met $1443s$ pour la résolution des 23 instances restantes, ce qui est expliqué par le filtrage de 2611 valeurs, permettant d'évaluer 460024 nœuds au total, contre 1046687 pour dual. Ceci montre encore une fois l'efficacité de la détection de la substituabilité pour les instances Taillard, et permet de faire le lien, comme pour l'approche statique entre valeurs filtrées et réduction de l'arbre de recherche.

Contrairement à l'approche statique, l'approche dynamique montre de meilleurs résultats en temps qu'un solveur dual classique, même si le nombre d'instances résolues reste le même. Aussi, la non-résolution de quelques instances par notre approche comparé à l'approche classique est due, dans le cas général, à la grande taille de l'instance ($fapp$ avec 2500 variables de taille maximum 360) ou le temps de résolution proche de $1200s$ pour l'approche classique, pour lequel le procédé de détection automatique pénalise notre approche.

5 Conclusion et perspectives

Dans ce papier, une généralisation de la substituabilité au voisinage, une forme de symétrie affaiblie, a été proposée. Cette généralisation originale est obtenue par substitution de la notion de supports de manière syntaxique par une mesure sémantique de la propagation des contraintes définie par l'état des variables dans le voisinage d'une variable affectée. Pour ceci, un prétraitement intégré à l'algorithme SAC a été proposé en tant qu'approche statique. De plus, puisque l'affectation et le maintien de la cohérence d'arc sont des opérations de base des algorithmes de type MAC, une adaptation dynamique a également été proposée. Les résultats expérimentaux sur les problèmes binaires de nos deux approches sont intéressantes par plusieurs aspects. Même si l'approche statique n'est pas rentable en temps, celle-ci se montre d'une grande efficacité de filtrage, ce qui permet la résolution d'instances que l'approche classique ne résoud pas. L'approche dynamique montre des résultats en temps intéressants, même si le nombre de problèmes sur lequel les valeurs substituables ont été détectées est faible. Enfin, certaines classes de problèmes pour lesquelles la détection des valeurs substituables s'avère efficace et pertinente peuvent être extraites et mettent évidence l'impact, positif et négatif, de la détection de la substituabilité au voisinage.

Les perspectives à la suite de ces travaux sont nombreuses. Outre la généralisation et l'amélioration de nos approches en pratique, l'utilisation d'autres formes de propagation des contraintes est envisagée. L'étude de l'impact d'un élagage sur la pondération d'heuristiques est envisagé pour tenter d'avoir une complémentarité. Il nous semble aussi intéressant d'exploiter notre approche pour établir des relations de dépendances entre les valeurs des variables.

Références

[1] Belaid Benhamou and Lakhdar Sais. Tractability through symmetries in propositional calculus. *Journal Automated Reasoning*, 12(1) :89–102, 1994.

[2] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy. Symmetry-breaking predicates for search problems. In *Principles of Knowledge Representation and Reasoning (KR'96)*, pages 148–159. 1996.

[3] Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *AAAI*, pages 227–233, 1991.

[4] Ian P. Gent and Barbara Smith. Symmetry breaking during search in constraint programming. In *Proceedings ECAI'2000*, pages 599–603, 2000.

[5] Christophe Lecoutre and Sébastien Tabary. Abscon 109 : a generic csp solver. In *2nd International CSP Solver Competition, held with CP'2006*, pages 55–63, 2007.

[6] J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Methodologies for Intelligent Systems : Proc. of the 7th International Symposium ISMIS-93*, pages 350–361. 1993.