Algorithme de décomposition de domaine pour la satisfaction et l'optimisation de contraintes

Wady Naanaa [1] **Maher Helaoui** [2] **Béchir Ayeb** [2]

[1] Faculté des sciences, Université de Monastir, Tunisie
[2] Pôle de Recherche en Informatique du Centre (PRINCE), Institut Supérieur d'Informatique et des Techniques de Communication, Route principale n°1 – 4011 Hammam Sousse
wnaanaa@planet.tn maher.helaoui@gmail.com ayeb_b@yahoo.com

Résumé

La notion de substituabilité directionnelle est une forme faible de la substituabilité de voisinage [6] qui a été proposée dans [17] pour améliorer la résolution de problèmes de satisfaction de contraintes (CSP) binaires. On part du fait que même si deux valeurs ne sont pas voisinage substituables, elles peuvent l'être si on restreint le voisinage en se référant à un ordre sur les variables. La substituabilité directionnelle permet de décomposer les domaines de valeurs des variables en de sous-ensembles de valeurs qui peuvent être essayées simultanément lors de la résolution du problème par un algorithme du type "*Backtracking*".

Dans le présent article, nous proposons deux extensions au travail présenté dans [17] :

- Tout d'abord, nous généralisons davantage la notion de substituabilité directionnelle en considérant, comme référence, une orientation du graphe d'inconsistance au lieu d'un ordre sur les variables.
- Ensuite, nous introduisons des conditions supplémentaires de substituabilité garantissant l'optimalité de la solution lors de la résolution de problèmes de satisfaction et d'optimisation de contraintes (CSOP).

Des résultats de simulation, sur plusieurs CSOPs modélisant des problèmes d'ordonnancement, montrent qu'une variante de l'algorithme du *Branch-and-Bound* qui exploite la substituabilité directionnelle est souvent plus efficace que l'algorithme original.

1 Introduction

Les problèmes de satisfaction de contraintes (CSPs) constituent un cadre simple et assez général pour modéliser divers problèmes combinatoires. Un CSP se définit par la donnée d'un ensemble de variables, chaque variable pouvant prendre une valeur parmi un ensemble de valeurs possibles appelé domaine. Puis, par un ensemble de contraintes exprimant des restrictions sur les combinaisons de valeurs permises. Résoudre un CSP revient à affecter aux variables, des valeurs de leurs domaines respectifs de telle sorte que toutes les contraintes soient satisfaites. Cette tâche est difficile du point de vue de la complexité puisqu'il s'agit de résoudre un problème NP-complet. En général, un CSP n'admet pas une solution unique. Pour les CSPs classiques, toutes les solutions sont supposées avoir la même "qualité". Ce qui fait que la résolution de tels problèmes se résume souvent à trouver la première solution. Cependant, dans plusieurs situations réelles telles que l'ordonnancement dans le secteur industriel, les solutions ne peuvent pas être considérées comme équivalentes dans le sens que certaines sont jugées meilleures que d'autres selon un critère bien déterminé. Il s'agit désormais de trouver la solution qui satisfait les contraintes et qui, en plus, optimise un critère mesurant la qualité des solutions. Les problèmes ainsi définis sont alors des problèmes de satisfaction de contraintes avec optimisation (CSOPs). Pour définir un CSOP et en plus des composantes définissant le CSP sous-jacent, on introduit une fonction coût qui permet d'exprimer des préférences entres les combinaisons de valeurs qui satisfont aux contraintes.

Dans cet article nous nous intéressons aux méthodes de résolution de CSOPs dites complètes : celles qui permettent de trouver une solution optimale si cette dernière existe. L'algorithme du *Branch-and-bound* [13] est sans doute l'algorithme complet le plus utilisé pour résoudre les CSOPs. C'est une procédure

d'exploration en profondeur d'abord munie d'une fonction heuristique servant à évaluer le coût de la solution complète que l'on peut espérer atteindre par l'extension de la solution partielle courante. Le rôle de la fonction heuristique est d'éviter l'exploration de branches de l'arbre de recherche qui ne mènent pas à des solutions meilleures que celles déjà trouvées.

Par ailleurs, la notion de substituabilité de voisinage [6] a été proposée et utilisée comme un moyen de filtrage des domaines de valeurs des variables des CSPs binaires. Plusieurs variantes de cette notion ont été proposées, parmi lesquelles la notion substituabilité directionnelle [17]. Cette dernière est une forme faible de la substituabilité de voisinage qui, au départ, visait à améliorer la résolution des CSPs binaires. On part de l'idée que même si deux valeurs ne sont pas voisinage substituables, elles peuvent l'être si on restreint le voisinage en se référant à un ordre sur les variables. La substituabilité directionnelle ne peut pas être utilisée comme un moyen de filtrage des domaines des variables comme c'est le cas pour la substituabilité de voisinage. Toutefois, on peut l'utiliser comme un moyen de décomposition des domaines des variables en de chaines de valeurs directionnellement substituables qui peuvent être considérées simultanément lors de la résolution des problèmes.
Dans le présent article, nous proposons deux extensions du travail présenté dans [17] :

- Nous généralisons davantage la notion de substituabilité directionnelle en considérant, comme référence, une orientation du graphe d'inconsistance au lieu d'un ordre sur les variables.
- Nous introduisons des conditions supplémentaires de substituabilité garantissant l'optimalité de la solution lors de la résolution de problème de satisfaction et d'optimisation de contraintes (CSOP).

Cet article est organisé comme suit : dans la section 2, nous donnons les définitions et les notations utilisées. Nous définissons par la suite (section 3), la notion de substituabilité directionnelle. On présente, dans la même section, une version de l'algorithme du Branch-and-Bound qui exploite la substituabilité directionnelle ainsi qu'un algorithme qui décompose les domaines de valeurs en de chaînes de valeurs directionnellement substituables. La section 4 présente un algorithme d'orientation du graphe d'inconsistance utilisée dans la détermination de la substituabilité directionnelle. Dans la section 5, nous reportons les résultats d'une expérimentation montrant l'apport de l'exploitation de la notion de substituabilité directionnelle pour la résolution de CSOPs binaires.

2 Définitions et notations

En plus des composantes qui définissent le problème de satisfaction de contrainte sous-jacent, un CSOP [23] fait intervenir une fonction coût qui permet de mesurer la qualité de chaque solution. Formellement, un CSOP peut être défini comme suit :

Définition 1 *Un problème de satisfaction et d'optimisation de contraintes (CSOP) est défini par un quadruplet (X, D, C, Z) tel que :*

1. *$X = \{x_1, ..., x_n\}$ est l'ensemble des variables.*
2. *$D = \{D_1, ..., D_n\}$ est l'ensemble des domaines de valeurs, D_k étant le domaine de x_k.*
3. *$C = \{C_1, ..., C_m\}$ est l'ensemble des contraintes. Chaque contrainte C_k implique un sous-ensemble de variables $Var(C_k) = x_{k_1}, ..., x_{k_r}$ appelé la portée de C_k et est définie par la relation, r-aire $Rel(C_k) \subseteq D_{k_1} \times D_{k_2} ... \times D_{k_r}$ contenant les r-uplets de valeurs respectant la contrainte C_k.*
4. *$Z : \prod_{i=1}^{n} D_i \longrightarrow \mathbb{R}$ est une fonction coût à minimiser.*

L'arité d'une contrainte est la taille de sa portée. L'arité d'un problème est l'arité maximale de ses contraintes. Dans cet article, nous nous limitons aux CSOPs binaires. Deux variables x_i et x_j reliées par une contrainte binaire, notée $C_{i,j}$, sont dites voisines. La valeur $a \in D_i$, dénotée aussi (x_i, a), est compatible avec $b \in D_j$ si $(a, b) \in Rel(C_{i,j})$. Dans ce cas, on dit que b est un support de a. Si chaque valeur du problème a, au moins, un support dans le domaine de chaque variable voisine alors le problème est dit arc-consistant. Le graphe d'inconsistance d'un CSOP binaire est un graphe simple dans lequel les sommets correspondent aux valeurs des variables et les arêtes relient les couples de sommets qui représentent des valeurs incompatibles. L'ensemble des valeurs incompatibles avec une valeur a d'une variable x_i, est défini par

$$N(x_i, a) = \{(x_j, b) \mid C_{i,j} \in C \text{ et } (a, b) \notin Rel(C_{i,j})\}$$

En se référant à [6], une valeur a d'une variable x_i est voisinage substituable (VS) à une valeur b de x_i si et seulement si $N(x_i, a) \subseteq N(x_i, b)$. Cette définition peut être étendue pour tenir compte de la fonction coût. Toutefois, cette dernière doit vérifier certaines propriétés pour que ceci soit possible. En effet, la fonction coût est une fonction globale (n-aire) puisqu'elle s'applique à toutes les variables du problème (voir définition 1) alors que la notion de substituabilité de voisinage est une notion locale. On propose donc que Z soit une fonction telle qu'il est possible de trouver une fonction coût unaire $z : \bigcup_{i=1}^{n} D_i \longrightarrow \mathbb{R}$ qui vérifie

$$\forall\ T \in D_1 \times \ldots \times D_n, \quad Z(T) = \oplus_{i=1}^{n} z(T_i) \qquad (1)$$

où $\oplus$ désigne un opérateur monotone sur $\mathbb{R}$ et T_i la valeur affectée à la variable x_i dans T. La substituabilité de voisinage pour les CSOPs binaires peut alors être définie de la manière suivante :

$$a \preceq b \quad \Leftrightarrow \quad \left\{ \begin{array}{l} N(x_i, a) \subseteq N(x_i, b) \quad \text{et} \\ z(x_i, a) \leq z(x_i, b) \end{array} \right.$$

Il en découle qu'à partir de toute solution dans laquelle x_i a pour valeur b, on peut déduire une solution de qualité, au moins, égale si l'on substitue a à b. Par conséquent, b peut être supprimée du domaine de x_i sans que l'on perde toutes les solutions optimales du problème.

3 Substituabilité directionnelle (DS)

3.1 Définitions et propriétés

La substituabilité directionnelle [17] est une forme faible de substituabilité de voisinage. Au départ, cette notion a été définie en se référant à un ordre total sur les variables du problème. Dans le présent article, on généralise la notion de substituabilité directionnelle en utilisant, comme référence, une orientation du graphe d'inconsistance du CSOP. Formellement, une orientation Λ du graphe d'inconsistance d'un CSOP est une affectation d'une direction à chaque arête $\{a, b\}$ du graphe donnant lieu, ou bien à l'arc (a, b) ou bien à l'arc (b, a). Etant donnée une orientation Λ du graphe d'inconsistance d'un CSOP, on définit l'ensemble des conflits directionnels d'une valeur a d'une variable x_i par rapport à Λ de la manière suivante :

$$\begin{array}{rcl} \vec{N}(x_i, a) & = & \{(x_j, b) \mid\ C_{i,j} \in C,\ (a, b) \notin Rel(C_{i,j}) \text{ et} \\ & & (a, b) \in\ \Lambda\} \end{array} \qquad (2)$$

La substituabilité directionnelle se définit alors comme suit :

Définition 2 *Soit $\mathcal{P} = (X, D, C, Z)$ un CSOP binaire et a et b deux valeurs d'une variable x_i de $\mathcal{P}$. a est dite directionnellement substituable à b par rapport à une orientation Λ du graphe d'inconsistance de $\mathcal{P}$, (notation : $a \preceq_\Lambda^{\mathcal{P}} b$) si et seulement si*

$$\vec{N}(x_i, a) \subseteq \vec{N}(x_i, b) \quad et \quad z(x_i, a) \leq z(x_i, b) \qquad (3)$$

Dans ce qui suit, on utilisera la notation $\preceq_\Lambda$ au lieu de $\preceq_\Lambda^{\mathcal{P}}$ si ceci n'entraine pas d'ambiguïté. La relation $\preceq_\Lambda$ définit un préordre sur le domaine de chaque variable. Ainsi, chaque domaine D_i peut être subdivisé en de sous-ensembles $D_i = D_{i,1} \cup \ldots \cup D_{i,s}$ tel que les éléments de chaque $D_{i,k},\ k = 1, ..., s$ sont tous deux à deux comparables. C'est-à-dire, que pour tout $a, b \in D_{i,k}$, on a $a \preceq_\Lambda b$ ou $b \preceq_\Lambda a$. Chaque $D_{i,k}$ est donc une chaine de valeurs totalement ordonnée par $\preceq_\Lambda$. Dans chaque chaine $D_{i,k}$, on peut distinguer le sous-ensemble des éléments minimaux.

$$\min(D_{i,k}) = \{a \in D_{i,k} \mid \forall\ b \in D_{i,k},\ a \preceq_\Lambda b\} \qquad (4)$$

La proposition suivante identifie une classe polynomiale de CSOPs.

Proposition 1 *Soit $\mathcal{P} = (X, D, C, Z)$ un CSOP binaire arc-consistant et soit Λ est une orientation du graphe d'inconsistance de $\mathcal{P}$. Si chacun des domaines de valeurs de $\mathcal{P}$ est une chaine non vide de valeurs directionnellement substituables par rapport à Λ alors une solution optimale de $\mathcal{P}$ peut être trouvée en un temps polynomial.*

Preuve : On montre qu'en sélectionnant un élément minimum de chaque domaine de valeurs, on obtient une solution optimale. Ce qui veut dire que tout n-uplet $T \in \min(D_1) \times \ldots \times \min(D_n)$ est une solution optimale. En se référant à (3) et au fait que la fonction z est calculable en temps polynomial, cette sélection peut être effectuée en temps polynomial.

Supposons que $T \in \min(D_1) \times \ldots \times \min(D_n)$ n'est pas une solution optimale. Ceci implique que T est inconsistant ou que $Z(T)$ n'est pas minimum.

Commençons par supposer que T est inconsistant. Donc T doit contenir, au moins, une paire de valeurs incompatibles. Soit $a \in \min(D_i)$ et $b \in \min(D_j)$ une telle paire. Puisque a et b sont incompatibles, on doit avoir $(a, b) \in \Lambda$ ou $(b, a) \in \Lambda$. Supposons, sans perte de généralité, que $(a, b) \in \Lambda$, (sinon on pourra raisonner sur b au lieu de a et obtenir le même résultat). Il s'ensuit que $b \in \vec{N}(x_i, a)$, et puisque $a \in \min(D_i)$ alors pour tout $a' \in D_i$, on doit avoir $b \in \vec{N}(x_i, a')$. Ce qui veut dire que b n'a pas de support dans D_i et donc que $\mathcal{P}$ n'est pas arc-consistant, d'où une contradiction.

Supposons, à présent que $Z(T)$ n'est pas minimum, donc, qu'il existe $T' \in D_1 \times \ldots \times D_n$ tel que $Z(T') < Z(T)$. Puisque les valeurs de la fonction Z sont obtenues à partir de celles de z en utilisant un opérateur monotone (voir équation (1)), il doit exister $x_i \in X$ tel que $T_i = (x_i, a), T_i' = (x_i, a')$ et $z(x_i, a') < z(x_i, a)$. Il en résulte que $a \notin \min(D_i)$, d'où une contradiction.

Nous nous intéressons aux méthodes de résolution de CSOPs dites complètes : celles qui permettent de trouver une solution optimale si cette dernière existe. L'algorithme du Branch-and-bound est l'algorithme complet le plus communément utilisé pour résoudre les CSOPs. Cet algorithme explore l'espace de recherche du problème à résoudre en effectuant une recherche arborescente en profondeur d'abord. Les problèmes considérés tout lelong d'un chemin de l'arbre de recherche sont des réductions du problème initial que l'on peut définir de la manière suivante :

Définition 3 *Un problème $\mathcal{P} = (X, D, C, Z)$ est une réduction d'un problème $\mathcal{P}' = (X', D', C', Z')$ (notation $\mathcal{P} \sqsubseteq \mathcal{P}'$) si et seulement si*
- $X = X'$,
- $D_i \subseteq D'_i$, pour tout $x_i \in X$,
- $C = \{C_{i,j} \mid C'_{i,j} \in C'$ et $Rel(C_{i,j}) = Rel(C'_{i,j}) \cap D_i \times D_j\}$,
- Z est la restriction de Z' à $D_1 \times \ldots \times D_n$.

Une propriété essentielle de la substituabilité directionnelle est qu'elle soit préservée quand on passe d'un problème à l'une de ses réductions. En effet, on a

Proposition 2 *Soient $\mathcal{P} = (X, D, C, Z)$ et $\mathcal{P}' = (X, D', C', Z)$ deux CSOPs binaires tels que $\mathcal{P} \sqsubseteq \mathcal{P}'$ et Λ, (resp. Λ') une orientation du graphe d'inconsistance de $\mathcal{P}$ (resp. de $\mathcal{P}'$) telle que $\Lambda \subseteq \Lambda'$. Alors, on a*

$$\forall\ x_i \in X,\ \forall\ a, b \in D_i \cap D'_i, \quad a \preceq^{\mathcal{P}'}_{\Lambda'} b \Rightarrow a \preceq^{\mathcal{P}}_{\Lambda} b$$

Preuve : Désignons par $\vec{N}^{\mathcal{P}}(x_i, a)$ l'ensemble des conflits directionnels de (x_i, a) dans $\mathcal{P}$ et par ΔD l'ensemble des valeurs de $\mathcal{P}'$ qui ne figurent pas dans $\mathcal{P}$. On a donc $\Delta D = \bigcup_{x_i \in X} D'_i - D_i$. Soient (x_i, a) et (x_i, b) deux valeurs quelconques disponibles dans $\mathcal{P}$ et $\mathcal{P}'$. Puisque $\mathcal{P} \sqsubseteq \mathcal{P}'$, on a

$$\vec{N}^{\mathcal{P}}(x_i, a) = \vec{N}^{\mathcal{P}'}(x_i, a) \ - \Delta D \tag{5}$$

De même

$$\vec{N}^{\mathcal{P}}(x_i, b) = \vec{N}^{\mathcal{P}'}(x_i, b) \ - \Delta D \tag{6}$$

D'autre part, en partant de $a \preceq^{\mathcal{P}'}_{\Lambda} b$, on déduit que $\vec{N}^{\mathcal{P}'}(x_i, a) - \Delta D \subseteq \vec{N}^{\mathcal{P}'}(x_i, b) - \Delta D$ et que $z(x_i, a) \leq z(x_i, b)$. D'après (5) et (6), on obtient $\vec{N}^{\mathcal{P}}(x_i, a) \subseteq \vec{N}^{\mathcal{P}}(x_i, b)$. D'où, $a \preceq^{\mathcal{P}}_{\Lambda} b$.

Une conséquence directe de la proposition 2 est que pour toute paire de CSOPs $\mathcal{P}$ et $\mathcal{P}'$ telles que $\mathcal{P} \sqsubseteq \mathcal{P}'$, si D'_i est une chaine de valeurs DS dans $\mathcal{P}'$ alors D_i est une chaine de valeurs DS dans $\mathcal{P}$.

3.2 Exemple

Considérons le problème Job-Shop classique décrit dans la figure 1. Il s'agit de planifier l'exécution de deux jobs J_1 et J_2 sur trois machines M_1, M_2 et M_3. Chaque job J_i se compose de trois tâches $T_{i,1}, T_{i,2}$ et $T_{i,3}$. Une tâche $T_{i,j}$ est dédiée à être exécutée sur une machine unique. Pour cet exemple, on suppose que la durée de toutes les tâches est fixée à une unité de temps, sauf $T_{2,2}$ qui consomme deux unités. Les job-shops font intervenir deux types de contrainte : des contraintes de précédence qui imposent que les tâches d'un même job soient exécutées les une après les autres ainsi que des contraintes de ressource qui empêchent qu'une machine exécute plus d'une tâche à la fois. Résoudre un tel problème revient à affecter un temps de début d'exécution à chacune des tâches de façon à satisfaire les contraintes de précédence et de ressource toute en minimisant le temps total d'exécution de toutes les tâches (makespan).

Une modélisation possible des problèmes job-shops en termes de CSOP binaire consiste à associer une variable du CSOP à chacune des tâches. Ainsi, pour notre exemple, on obtient un problème impliquant six variables $x_1, \ldots, x_6$. Les domaines de valeurs des variables représentent une discrétisation du temps maximum impartie à l'exécution de toutes les tâches. Pour le présent exemple, on suppose que les différentes tâches peuvent commencer à des instants représentés par les entiers de 0 à 4. Les contraintes impliquées sont des contraintes binaires de précédence et de ressource. On en trouve en tout sept contraintes (voir figure 1). La fonction coût à minimiser est définie par

$$Z(T) = \max_{1 \leq i \leq 6} \{a + durée(x_i) \mid T_i = (x_i, a)\}$$

où T désigne une instanciation de toutes les variables du CSOP. En tenant compte des contraintes de précédence, la fonction Z peut être calculée en prenant le maximum uniquement sur la paire $\{x_3, x_6\}$ au lieu de X. On en déduit une définition possible de la fonction z

$$z(x_i, a) = \begin{cases} a + durée(x_i) & \text{si } i \in \{3, 6\} \\ 0 & \text{sinon} \end{cases} \tag{7}$$

On obtient donc $Z(T) = \max_{i=1}^{6}(z(T_i))$, où max est bien un opérateur monotone. Après application d'un algorithme d'arc-consistance, on obtient le problème dont le graphe d'inconsistance est représenté dans la figure 2. Dans cette même figure on peut également voir une orientation du graphe d'inconsistance. En se référant à cette orientation, on obtient les ensembles des conflits directionnels donnés dans le tableau 1. En examinant ce tableau, on constate que

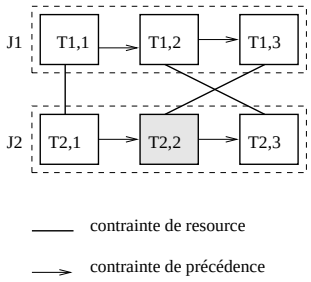

FIGURE 1 – Un problème du type job-shop comportant deux jobs et six tâches. Toutes les tâches sont supposées avoir une durée d'une unité de temps, sauf $T_{2,2}$ qui en consomme deux.

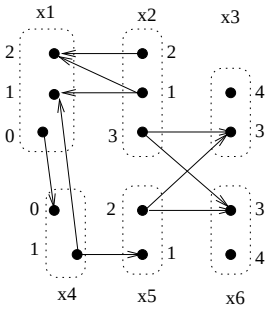

FIGURE 2 – Graphe d'inconsistance du job-shop décrit dans la figure 1 après application d'un algorithme d'arc-consistance. On peut également voir une orientation du graphe d'inconsistance.

tous les domaines de valeurs sont des chaines de valeurs DS sauf D_2 qui est composé de deux chaines : $\{(x_2,1),(x_2,2)\}$ et $\{(x_2,3)\}$. En réduisant le domaine de x_2 à $\{(x_2,1),(x_2,2)\}$ puis à $\{(x_2,3)\}$, on obtient deux sous-problèmes qui, d'après la proposition 2, ne contiennent que des chaines de valeurs DS comme domaine. D'après la proposition 1, ces deux sous-problèmes peuvent être résolus en un temps polynomial.

3.3 Exploitation de la Substituabilité Directionnelle

La substituabilité directionnelle peut être utilisée pour améliorer la résolution de CSOPs binaires et ceci en l'intégrant à l'algorithme du Branch-and-Bound pour obtenir l'algorithme BAB–DS (voir fonction 1). Pour avoir une méthode de résolution plus efficace, BAB–DS intègre également une procédure qui permet de maintenir l'arc-consistance durant la recherche comme dans l'algorithme de résolution de CSP MAC [19].

BAB–DS prend comme paramètres le CSOP à

	0	1	2	3	4
x_1	$(x_4,0)$	$\emptyset$	$\emptyset$		
x_2		$(x_1,1)$ $(x_1,2)$	$(x_1,2)$	$(x_3,3)$ $(x_6,3)$	
x_3				$\emptyset$	$\emptyset$
x_4	$\emptyset$	$(x_1,1)$ $(x_5,1)$			
x_5		$\emptyset$	$(x_3,3)$ $(x_6,3)$		
x_6				$\emptyset$	$\emptyset$

TABLE 1 – Conflits directionnels des valeurs arc-consistantes du CSOP décrit dans la figure 2.

résoudre (X,D,C,Z), (qui est supposé être arc-consistant), une orientation du graphe d'inconsistance du problème Λ, l'ensemble des variables non encore instanciées Y et la meilleure solution courante I^* et procède comme suit : Il sélectionne une variable non encore instanciée, calcule la partition de son domaine de valeurs en des chaines de valeurs DS et décompose le problème courant en deux sous-problèmes. Le premier sous-problème est obtenu en réduisant le domaine de valeurs de la variable courante à une chaine de valeurs DS (notée $D_{i,k}$ dans le pseudo-code). L'arc-consistance du problème résultant est alors restauré en utilisant un algorithme d'arc-consistance. Puis, un appel récursif est effectué pour considérer les variables restantes. Cet appel permet d'obtenir la meilleure solution du sous-problème qui limite les valeurs possibles de la variable courante aux éléments de $D_{i,k}$. Ensuite, un processus de restauration des domaines de valeurs est effectué et $D_{i,k}$ est éliminée du domaine de la variable courante. Après restauration de l'arc-consistance du problème résultant, l'algorithme effectue un deuxième appel récursif pour considérer les autres chaines de valeurs DS. Les deux appels récursifs ne sont effectués que si une fonction heuristique (h) indique qu'il est possible d'obtenir des solutions de qualité meilleures que celles déjà trouvées. Si l'algorithme réussit à instancier toutes les variables alors il exécute un processus polynomial (ligne 4) pour extraire la meilleure solution à partir des chaines sélectionnées.

Fonction 1 BAB–DS$((X,D,C,Z),\Lambda,Y,I^*)$

1 : **si** $Y=\emptyset$ **alors**
2 : retourner(min(D))
3 : **sinon**
4 : $x_i \leftarrow$ Sélect(Y)
5 : $D_i \leftarrow$ DS-Partition$(D_i,\Lambda,(X,D,C,Z))$
6 : $D_{i,k} \leftarrow$ Select (D_i)
7 : $D_i \leftarrow D_{i,k}$
8 : AC(X,D,C)
9 : **si** $\emptyset \notin D$ **et** $h(D) < Z(I)$ **alors**

10 : $I \leftarrow$BAB–DS$(Y - \{x_i\}, (X, D, C, Z), \Lambda, I^*)$
11 : **si** $Z(I) < Z(I^*)$ **alors** $I^* \leftarrow I$
12 : Restaurer (D)
13 : $D_i \leftarrow D_i$ - $D_{i,k}$
14 : AC(X, D, C)
15 : **si** $\emptyset \notin D$ **et** $h(D) < Z(I^*)$ **alors**
16 : $I \leftarrow$BAB–DS$(Y - \{x_i\}, (X, D, C, Z), \Lambda, I^*)$
17 : **si** $Z(I) < Z(I^*)$ **alors** $I^* \leftarrow I$
18 : Restaurer(D)
19 : retourner(I^*)

3.4 Algorithme de DS partition

La relation $\preceq_\Lambda$ définit un préordre sur le domaine de chaque variable du problème. Ce préordre peut être utilisé pour partitionner les domaines de valeurs des variables en de chaines de valeurs DS. En effet, à partir de $\preceq_\Lambda$, on définit, tout d'abord, la relation $\sim_\Lambda$ telle que $a \sim_\Lambda b$ si et seulement si $a \preceq_\Lambda b$ et $b \preceq_\Lambda a$. On peut facilement vérifier que $\sim_\Lambda$ est une relation d'équivalence. $\preceq_\Lambda$ induit un ordre partiel $\leq_\Lambda$ sur l'ensemble $D_i \backslash \sim_\Lambda$ des classes d'équivalence de $\sim_\Lambda$ tel que $[a] \leq_\Lambda [b]$ si et seulement si $a \preceq_\Lambda b$, où $[a]$ désigne la classe d'équivalence de la valeur a.

En général, étant donné un ensemble partiellement ordonné E, on peut avoir plusieurs partitions de E en chaines d'éléments totalement ordonnés. En théorie des ensembles, la partition optimale en chaines d'un ensemble partiellement ordonné est connue sous le nom de partition en chaines de Dilworth (DCP) [5]. C'est une partition qui contient le nombre minimum de chaines parmi toutes les partitions possibles. La taille d'une telle partition (i.e., le nombre de chaines de la partition), définit la largeur de l'ordre partiel. Le problème qui consiste à trouver la DCP d'un ensemble partiellement ordonné peut être résolu en temps polynomial en l'exprimant comme un problème de recherche d'un couplage maximum dans un graphe bipartite [9].

Motivé par le fait qu'à chaque noeud de l'arbre de recherche, l'algorithme de résolution aura autant de choix que de chaines dans une DCP du domaine de la variable courante, nous proposons de calculer une DCP à chaque noeud de l'arbre de recherche. Par cette stratégie, on cherche à minimiser la taille de l'arbre de recherche que l'algorithme aura à explorer.

La première étape du calcul de la DCP (voir la fonction 2) consiste à déterminer la relation $\sim_\Lambda$. Ceci peut être accompli en O(nd^2) étapes en utilisant l'algorithme décrit dans [6] ou celui proposé dans [17], n et d étant respectivement, le nombre de variables du problème et la taille maximale des domaines de valeurs. On en déduit les classes d'équivalences $D_i/\sim_\Lambda$ en $O(d)$ étapes. L'étape la plus couteuse est celle du calcul de la relation $\leq_\Lambda$. Au pire des cas, $d(d-1)/2$ tests d'inclusion entre des paires d'ensembles de conflits directionnels sont nécessaires. Chaque test d'inclusion peut être accompli en $O(nd)$ puisque chaque ensemble de conflit directionnel contient au maximum $d(n-1)$ éléments. D'où une complexité de O(nd^3) pour cette étape. Ensuite, l'algorithme construit un graphe bipartite $G = (V, U, E)$ tel que $V = U = D_i/\sim_\Lambda$ et l'ensemble des arêtes E contient une arête $([a], [b])$ si et seulement si $[a] \leq_\Lambda [b]$. La construction de G nécessite O(d^2) étapes. Un algorithme de couplage maximum est alors appliqué à G. On utilise l'algorithme décrit dans [9] qui s'exécute en O$(d^{2.5})$ étapes. Les chaines de la DCP sont extraites du couplage maximum en incluant les éléments de $[a]$ et ceux de $[b]$ dans une même chaine chaque fois que l'arête $([a], [b])$ fait partie du couplage maximum. Ceci demande O(d^2) étapes. D'où une complexité totale de O(nd^3).

Fonction 2 DS–Partition$(D_i, \Lambda, (X, D, C, Z))$

1 : $\sim_\Lambda \longleftarrow$ DirInterchangeable$(D_i, \Lambda, (X, D, C, Z))$
2 : $D_i/\sim_\Lambda \longleftarrow$ ExtractEqClass$(\sim_\Lambda, D_i, (X, D, C, Z))$
3 : $\leq_\Lambda \longleftarrow$ DirSubstituable$(D_i/\sim_\Lambda, (X, D, C, Z))$
4 : $G \longleftarrow$ BipartiteGraph$(D_i/\sim_\Lambda, \leq_\Lambda)$
5 : $M \longleftarrow$ CouplageMaximum(G)
6 : $\mathcal{D}_i \longleftarrow$ ExtractChaines$(D_i, \sim_\Lambda, M)$
7 : return$(\mathcal{D}_i)$

3.5 Exemple (suite)

Reprenons l'exemple du paragraphe 3.2. Comme on l'a déjà constaté, après application d'un algorithme d'arc-consistance, tous les domaines de valeurs se trouvent réduits à des chaines de valeurs DS sauf D_2 qui est composé de deux chaines : $\{(x_2, 1), (x_2, 2)\}$ et $\{(x_2, 3)\}$. L'application de l'algorithme BAB–DS à cet exemple se résume à instancier les variables x_1, x_3, x_4, x_5 et x_6 par les seules chaines qui constituent leurs domaines respectifs. Pour x_2, il y a deux instanciations possibles qui correspondent aux deux chaines données ci-dessus. Commençons par réduire D_2 à la chaine $\{(x_2, 1), (x_2, 2)\}$. A ce stade, tous les domaines sont des chaines de valeurs DS. La restauration de l'arc-consistance (ligne 8) ne change rien à ce fait d'après la proposition 2. Le graphe d'inconsistance du problème obtenu est illustré dans la figure 3-(a). Par la suite, l'algorithme choisit un élément minimum de chacun des domaines (ligne 2). D'après le tableau 1 et la figure 3-(a), le choix est unique pour chacune des variables et l'on obtient l'instanciation complète $(1, 2, 3, 0, 1, 3)$ qui est bien une solution qui satisfait toutes les contraintes et qui a un coût égal à 4 unités de temps. Enfin, BAB–DS réduit D_2 à $\{(x_2, 3)\}$.

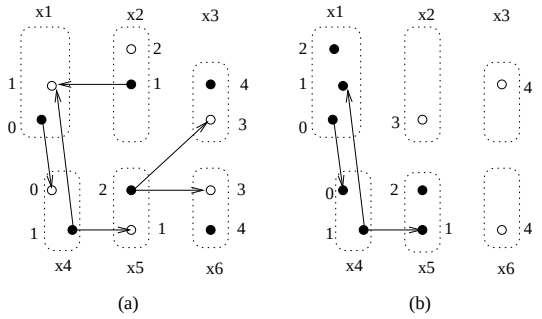

FIGURE 3 – (a) Graphe d'inconsistance du CSOP associé au job-shop décrit dans la figure 1 après réduction de D_2 à la chaine $\{(x_2, 1), (x_2, 2)\}$ et application d'un algorithme d'arc-consistance. (b) Graphe d'inconsistance du même CSOP après réduction de D_2 à la chaine $\{(x_2, 3)\}$ et application d'un algorithme d'arc-consistance.

En appliquant l'algorithme d'arc-consistance, puis la fonction heuristique h sur les domaines ainsi réduits, on déduit qu'il n'est pas possible de trouver une solution meilleure que celle déjà trouvée. En effet, les domaines de x_3 et de x_6 sont tous les deux réduits à la seule valeur 4 (voir figure 3-(b)). Ce qui veut dire que la meilleure solution qu'on peut espérer trouver a un coût de 5 unités de temps.

4 Orientation du graphe d'inconsistance

L'orientation du graphe d'inconsistance utilisée comme référence (voir équation 2) pour déterminer les valeurs DS a un grand impact sur l'occurrence des valeurs DS. Les meilleures résultats ont été obtenus lorsqu'on s'est référé à une orientation qui favorise l'occurrence des valeurs DS à des niveaux proches de la racine de l'arbre de recherche. Une telle orientation est obtenue dynamiquement en utilisant un algorithme glouton (voir fonction 3). Au départ du processus de recherche, l'orientation Λ est vide. A chaque étape, l'algorithme d'orientation ne considère que les arêtes qui relient les valeurs de la variable courante à ceux des variables déjà instanciées. Soit a une valeur quelconque de la variable courante qui est incompatible avec une valeur b d'une variable déjà instanciée. L'arête $\{a, b\}$ donnera lieu à l'arc (a, b), qui sera ajouté à Λ.

Cette stratégie favorise l'occurrence des valeurs DS à des niveaux proches de la racine. En effet, rappelons que seulement les arcs sortants d'une valeur sont pris en considération lors du calcul des conflits directionnels de cette valeur. En ne considérant que les arêtes qui vont vers les valeurs des variables instanciées, on augmente les chances des valeurs des variables traitées à des niveaux proches de la racine à être DS les une aux autres. La raison de ceci est que, proche de racine, il y a peu de variables instanciées. Le fait de considérer de larges chaines de valeurs DS à des niveaux peu profonds de l'arbre de recherche est une stratégie souvent avantageuse car, à ce stade, l'heuristique d'ordre des valeurs ne dispose pas d'assez d'information pour faire des choix de valeurs corrects. En prenant des décisions portant sur plusieurs valeurs, on peut éviter des erreurs qui, plus elles sont commises tôt, plus elles sont couteuses en temps d'exploration.

Fonction 3 Orientation$(x_i, \Lambda, Y, (X, D, C, Z))$

1 : **pour tout** $x_j \in X - Y \mid C_{i,j} \in C$ **faire**
2 : **pour tout** $a \in D_i$ **faire**
3 : **pour tout** $b \in D_j \mid (a, b) \notin Rel(C_{i,j})$ **faire**
4 : $\Lambda \longleftarrow \Lambda \cup \{(a, b)\}$
5 : Retourer(Λ)

5 Résultats expérimentaux

L'expérimentation porte sur la comparaison des performances de l'algorithme du Branch-and-Bound classique et de BAB–DS sur des problèmes du type job-shop [1] qui ont été modélisés comme indiqué dans le paragraphe 3.2. L'algorithme d'arc-consistance employé est AC-3 [16]. L'heuristique d'ordre de variables est *min-domain/wdeg* [2], Pour l'ordre des valeurs, nous avons testé deux heuristiques : *min. conflit* qui privilégie les valeurs qui ont le moins de conflits [7] et *min. valeur* qui tient compte de la fonction coût en favorisant les plus petites valeurs. La fonction z, qui est indispensable pour définir la substituabilité directionnelle, est déduite de Z de manière analogue à celle spécifiée par l'équation (7). Les critères d'évaluation des performances sont le temps de calcul et la qualité de la meilleure solution trouvée. Les deux algorithmes ont été implémentés en C++ et exécutés sur un PC à 2 GHZ de fréquence et une mémoire vive de 4GB. Les problèmes test utilisés sont ceux référencés par JS-TAILLARD-15 disponibles sur [14]. Ce sont dix instances de job-shops comportant 15 jobs et 15 machines, (donc 225 tâches), générés selon le modèle décrit dans [22]. Pour ces instances de job-shop, la discrétisation du temps maximum impartie pour l'exécution de toutes les tâches donne plus de mille valeurs. Pour obtenir des solutions en des temps d'exécution raisonnables pour des CSOPs impliquant de si large domaines de valeurs, on a élargi davantage les domaines de valeurs pour obtenir des instances plus fa-

ciles. Ainsi, on a multiplié les tailles des domaines originaux par un facteur de 105%, 104% puis 103% pour obtenir des instances qui vont des plus faciles aux plus difficiles. De plus, on a fait recours à une recherche par divergence limitée [8], pour les deux algorithmes.

Les résultats obtenus (voir figure 4) montrons clairement que BAB–DS a trouvé des solutions pour plus d'instances que BAB. Pour les instances auxquelles les deux algorithmes ont réussit à trouver des solutions, on constate que dans la plupart des cas, ou bien que la solution trouvée par BAB–DS a un coût meilleur que celle trouvée par BAB, ou bien que BAB–DS trouve une solution ayant le même coût mais en moins de temps. Avec l'heuristique min. conflit, sur les trente instances considérées, il y a uniquement trois instances (tai15-104-4, tai15-104-7 et tai15-103-7) sur lesquelles, BAB a été meilleur que BAB–DS. Avec l'heuristique min. valeur, BAB–DS a donné des résultats meilleures sur toutes les instances sauf l'instance tai15-104-9. Enfin, la supériorité de BAB–DS ne semble dépendre ni de la difficulté des instances ni de l'heuristique d'ordre des valeurs utilisée puisque BAB–DS est globalement meilleur sur les trois niveaux de difficulté.

6 Travaux Connexes

Plusieurs travaux ont porté sur l'exploitation dynamique de l'interchangeabilité de voisinage qui est un cas particulier de la substituabilité de voisinage [10, 15, 21]. Les algorithmes proposés détectent des sous-ensembles de valeurs voisinage interchangeables, dont le produit cartésien fournit une représentation compacte de l'ensemble de solutions du CSP. Cette représentation des solutions par produit cartésien est très utilisée pour le calcul de toutes les solutions. Dans [4], les auteurs montrent que les algorithmes s'appuyant sur la représentation par produit cartésien sont aussi efficaces pour la recherche de la première solution d'instances difficiles de CSP.

Toutefois, l'efficacité de ces méthodes reste tributaire de l'occurrence de l'interchangeabilité de voisinage. Cette dernière est plutôt rare quand il s'agit de situations réelles. Pour cette raison, plusieurs variantes de l'interchangeabilité de voisinage ont été proposées. Dans [24], les auteurs proposent la notion d'interchangeabilité conditionnelle, qui vise à détecter plus de valeurs voisinage interchangeables. Une condition est une restriction du domaine des variables voisines qui permet de capturer l'interchangeabilité dans des situations où l'interchangeabilité de voisinage ne s'applique pas. Bowen et Likitvivatanavong ont introduit le concept de transmutation de domaine [3]. Leur approche consiste à regrouper plusieurs valeurs afin d'exploiter plus intensivement l'interchangeabilité. Les auteurs ont reporté que leur approche est particulièrement avantageuse dans la recherche de toutes les solutions d'un CSP. Le filtrage par interchangeabilité de voisinage a été aussi appliqué aux CSPs non binaires [12]. Dans [18], les auteurs ont généralisé la définition d'interchangeabilité en vue de l'appliquer aux soft CSPs. Ils ont introduit deux relaxations en s'appuyant sur les notions de dégradation et de seuil. Ces deux formes d'interchangeabilités sont respectivement notées ${}^{\delta}$interchangeabilité de voisinage (${}^{\delta}$NI) et ${}_{\alpha}$interchangeabilité voisinage (${}_{\alpha}$NI). Il a été expérimentalement montré que les valeurs ${}^{\delta}$NI et ${}_{\alpha}$NI sont fréquemment rencontrées lors de la résolution de CSPs flous et de CSPs pondérés. Enfin, la notion de substituabilité a été également définie pour les contraintes où des tuples redondants sont détectés et supprimés des relations définissant les contraintes [11], .

Toutes les approches citées ci-dessus utilisent diverses formes d'interchangeabilité ou de substituabilié pour déterminer les valeurs ou les tuples qu'on peut supprimer sans perdre toutes les solutions (optimales) du problème. Ces méthodes peuvent donc être classées comme des méthodes de filtrage. Dans notre cas, la substituabilité directionnelle ne permet pas d'éliminer les valeurs ou les tuples redondants, mais permet, à un algorithme de recherche arborescente, d'effectuer des branchements sur plusieurs valeurs à la fois. La substituabilité directionnelle est donc une méthode de décomposition de domaine.

7 Conclusion

Dans cet article, nous avons proposé deux extensions à la notion de substituabilité directionnelle présentée par l'un des auteurs dans [17]. Tout d'abord, nous avons généralisé davantage la substituabilité directionnelle en considérant, comme référence, une orientation du graphe d'inconsistance du problème au lieu d'un ordre sur les variables du problème. Puis, nous avons introduit des conditions supplémentaires de substituabilité garantissant l'optimalité de la solution lors de la résolution de problème de satisfaction et d'optimisation de contraintes (CSOP).
Les résultats de simulation sur plusieurs CSOPs qui codent des problèmes d'ordonnancement du type jobshop ont montré qu'une variante de l'algorithme du Branch-and-Bound exploitant la substituabilité directionnelle est souvent plus efficace que l'algorithme original.

Une extension possible de ce travail consiste à proposer une formulation encore plus générale de la substituabilité directionnelle afin de pouvoir l'appliquer aux Soft CSPs [20] qui est un formalisme offrant une plus grande capacité à exprimer des problèmes réels.

Références

[1] Applegate, D., Cook, W. A computational study of the job-shop scheduling problem, *ORSA Journal on Computing* 3 : 149-156. 1991

[2] Boussemart, F., Hemery, F., Lecoutre, C., Sais, L. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004

[3] Bowen, J., Likitvivatanavong, C. Domain transmutation in constraint satisfaction problems. In *Proceedings of AAAI-04*, 2004

[4] Choueiry, B. Y., Davis, A. M. Dynamic Bundling : less effort for more solutions. In *Proceedings of the* 5^{th} *International Symposium on Abstraction, Reformulation and Approximation, Lecture Notes in Computer Science*, pages 64–82, volume 2371, Springer-Verlag, 2002

[5] Dilworth, R. P. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51 : 161-166. 1950

[6] Freuder, E. C. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI-91*, pages 227–233, 1991

[7] Frost, D., Dechter, R. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 572–578, 1995

[8] Harvey, W., Ginsberg, M. Limited discrepancy search. In *Proceedings of the 14th International Joint Conference On Artificial Intelligence*, pages 607–613. 1995

[9] Hopcroft, J. E., Karp, R. M. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM Journal on Computing*, 2, 4 : 225–231, 1993

[10] Hubbe, P. D., Freuder, E. C. An efficient cross product representation of the constraint satisfaction problem search space. In *Proceedings of the* 10^{th} *National Conference on Artificial Intelligence*, pages 421–427. San Jose, California, USA, 1992

[11] Jeavons, P. G., Cohen, D. A., Cooper, M. C. A substitution operation for constraints. In *Proceedings of the Second International Workshop on Principal and Proctice of Constraint Programming (PPCP94)*, pages 1–9, Lecture Notes in Computer Science 874 Springer-Verlag, 1994

[12] Lal, A., Choueiry, B.Y., Freuder, E.C. Neighborhood interchangeability and dynamic bundling for non-binary finite CSPs. In *Proceedings of AAAI-05*, pages 397–404, 2005

[13] Lawler, E. L., et Wood, D. E. Branch-and-bound methods : A survey. *Operations Research*, 14 :699-719 1966

[14] Lecoutre, C. Toward Benchmarks 2.1. XML representation of CSP instances. *http ://www.cril.univ-artois.fr/∼lecoutre/research/benchmarks/benchmarks.html*, 2007

[15] Lesaint, D. Maximal sets of solutions for constraint satisfaction problems. In *Proceedings of the* 11^{th} *European Conference on Artificial Intelligence*, pages 110–114, Amsterdam, The Netherlands, 1994

[16] Mackworth., A., K. consistancy in networks of relations. *Artificial Intelligence*, 8 : 99–118, 1977

[17] Naanaa, W. Directional interchangeability for enhancing CSP solving. *CPAIOR 2007*, pages 200–213, Hentenryck and Wolsey Eds. LNCS 4510, 2007

[18] Neagu, N., Bistarelli, S., Faltings, B. Experimental Evaluation of Interchangeability in Soft CSPs. *International Workshop on Constraint Solving and Constraint Logic Programming (CSCLP 2003)*, pages 140–153. 2003

[19] Sabin, D., Freuder, E.C. Contradicting conventional wisdom in constraint satisfaction. *Proceedings of the 11th European Conference on Artificial Intelligence*, 1994

[20] Schiex, T., Fargier, H., Verfaillie, G. Valued constraint satisfaction problems : hard and easy problems. In *Proceedings of the* 14^th *IJCAI*, pages 631–637, Montréal, Canada, 1995

[21] Silaghi, M. C., Sam-Haroud, D., Faltings, B. V. Ways of maintaining arc-consistency in search using the Cartesian representation *New Trends in Constraints, Joint ERCIM/Compulog Net Workshop*, pages 173–187, Springer-Verlag, 2000

[22] Taillard, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64 : 278–285, 1993

[23] Tsang, E. Foundations of constraint satisfaction. *Published by Academic Press Limited in UK* 24-28 Oval Road, London NW1 7DX USA : Sandiego, CA 92101 ISBN 0-12-701610-4. 1993

[24] Zhang, Y., Freuder, E.C. Conditional interchangeability and substitutability. In *Proceedings of Fourth International Workshop on Symmetry and Constraint Satisfaction Problems (SymCon04)*, Toronto, 2004

	Min. conflit				Min. valeur			
	temps		coût		temps		coût	
inst.	BAB	BAB-DS	BAB	BAB-DS	BAB	BAB-DS	BAB	BAB-DS
105-0	7065	651	1308	1308	–	225	–	1290
105-1	90	3847	1326	1325	–	1573	–	1304
105-2	11521	748	1295	1295	–	1173	–	1285
105-3	–	3743	–	1239	–	6193	–	1234
105-4	–	5503	–	1297	4491	12943	1291	1285
105-5	–	2532	–	1308	–	2249	–	1309
105-6	105	6056	1297	1296	2292	9559	1284	1286
105-7	–	376	–	1282	–	523	–	1282
105-8	3406	74	1353	1353	–	2541	–	1343
105-9	3051	4681	1334	1333	739	13850	1334	1316

	Min. conflit				Min. valeur			
	temps		coût		temps		coût	
inst.	BAB	BAB-DS	BAB	BAB-DS	BAB	BAB-DS	BAB	BAB-DS
104-0	–	4526	–	1296	12157	3366	1292	1279
104-1	–	460	–	1313	–	12147	–	1302
104-2	–	88	–	1283	–	13812	–	1269
104-3	646	2716	1228	1227	–	11080	–	1228
104-4	1500	12942	1285	1285	–	10504	–	1281
104-5	–	643	–	1297	–	–	–	–
104-6	–	1136	–	1284	–	766	–	1282
104-7	2606	14028	1270	1270	–	–	–	–
104-8	61	539	1341	1340	–	4137	–	1340
104-9	99	79	1321	1321	739	–	1318	–

	Min. conflit				Min. valeur			
	temps		coût		temps		coût	
inst.	BAB	BAB-DS	BAB	BAB-DS	BAB	BAB-DS	BAB	BAB-DS
103-0	–	122	–	1284	–	–	–	–
103-1	–	885	–	1301	–	3813	–	1284
103-2	–	423	–	1271	–	389	–	1262
103-3	–	–	–	–	–	9734	–	1213
103-4	–	–	–	–	–	218	–	1270
103-5	–	–	–	–	–	–	–	–
103-6	–	1274	–	1273	–	–	–	–
103-7	2513	–	1259	–	–	–	–	–
103-8	–	–	–	–	–	132	–	1327
103-9	–	454	–	1309	–	8926	–	1287

FIGURE 4 – Résultats obtenus par BAB et BAB-DS sur les instances js-taillard-15-(105 à 103) avec deux heuristiques d'ordre des valeurs : min. conflit et min. valeur. Le temps CPU en secondes et le coût de la meilleure solution trouvée sont indiqués. Pour le temps CPU, on indique les temps auxquels ont été trouvées les meilleures solutions. Le temps d'exécution a été limité à 4 heures par instance.