Inférence d'Inconsistances pour la Résolution de MAX-N CSP

Fayçal Djerourou, Hachemi Bennaceur

Inférence d'inconsistances pour la résolution du Max-CSP

Fayçal Djerourou **Hachemi Bennaceur**

LIPN-UMR CNRS 7030- Institut Galiliée, Université de Paris Nord,
99, avenue Jean-Baptiste Clément, 93430, Villetaneuse, France.
{Faycal.Djerourou,Hachemi.Bennaceur}@lipn.univ-paris13.fr

Résumé

Le présent article porte sur la résolution exacte du problème de satisfaction maximale de contraintes (Max-CSP). La résolution du Max-CSP est généralement effectuée à l'aide d'algorithmes de type *branch and bound*. L'efficacité de ces algorithmes dépend étroitement de la borne inférieure de la valeur du Max-CSP qu'ils utilisent au cours de la recherche. Les bornes inférieures pour le Max-CSP se basent, en général, sur le concept d'arc consistance (AC). Nous présentons ici un mécanisme d'inférences d'inconsistance entre valeurs dans le but d'augmenter les valeurs des bornes inférieures basées sur l'AC. Ceci est effectué en détectant les inconsistances ignorées par les compteurs d'arc-inconsistance dirigée *dac*. L'inférence d'inconsistances pour une valeur a d'une variable non instanciée i, consiste à rajouter à la contribution de a le nombre de variables futures dont les valeurs participant au calcul de la borne inférieure sont incohérentes avec a. Une implémentation de cette technique est présentée et testée sur des Max-CSP aléatoires.

Abstract

This paper deals with an exact algorithm designed for solving over-constrained problems. It is well known that the efficiency of a complete algorithm deeply depends on the quality of its lower bound. This lower bound should be as large as possible and as cheap to compute as possible. We propose a new technique which enhances the classical arc consistency lower bounds for Max-CSP. In addition to *inconsistency* and *directed arc consistency counts* of a value, we infer further inconsistencies resulting from inconsistencies between this value and minima values of unassigned variables. We will show in this paper that this idea can be generalized to the Weighted CSP framework (see section 4.2). The inferred inconsistencies technique is embedded in a forward checking algorithm, it is used to tight the lower bound during the search in order to filter future domains. This new algorithm, PFC-VRDAC (*PFC-Value inference Reversible Directed Arc Consistency*), is compared with some reference algorithms for solving Max-CSP. Obtained results on some random problems, show improvements in time and search effort.

1 Introduction

Le problème de satisfaction de contraintes (CSP) est donné par un ensemble de variables soumises à des contraintes. Chaque variable possède un domaine discrêt et fini. Le but est d'affecter à chacune des variables une valeur de son domaine afin de satisfaire l'ensemble des contraintes. Nous nous intéressons ici aux problème de satisfaction de contraintes *surcontraint* où la satisfaction de toutes les contraintes du CSP est impossible. Dans ce cas, le but est de chercher l'instanciation complète satisfaisant le maximum de contraintes (Max-CSP). Le Max-CSP est un problème d'optimisation. Dans [13], le CSP a été étendu vers un modèle plus général le *problème de satisfaction de contraintes valué* (VCSP). Ce modèle prend en compte les contraintes *molles* et rend possible l'expression des préférences entre solutions. En VCSP des *valuations* sont associées aux n-uplets et le coût d'une instanciation complète est calculé en agrégeant les valuations des n-uplets appartenant à cette instanciation. L'objectif en VCSP est de trouver l'instanciation complète ayant le coût minimal. Dans [6], le *Weighted* CSP, un cas particulier du VCSP est proposé. En WCSP les valuations sont des entiers positifs et l'opérateur d'agrégation est l'addition. Le VCSP et ses spécifications (WCSP et Max-CSP) possèdent plusieurs domaines d'application tels que l'*allocation de ressources*, la *bioinformatique*, le *raison-*

nement probabiliste et l'*ordonnancement*.

La résolution exacte du Max-CSP est généralement effectuée avec des méthodes suivant le schéma du *branch and bound*. Ces méthodes parcourent l'arbre de recherche des solutions en profondeur d'abord. Pour diriger la recherche, ils utilisent une borne inférieure du nombre minimal de contraintes violées. L'efficacité d'un branch and bound dépend étroitement de la qualité de sa borne, celle-ci doit être la plus proche possible de la valeur du problème et non coûteuse à calculer. Différentes bornes inférieures ont été définies pour le Max-CSP, elles sont basées sur le concept d'arc consistance (AC) [1, 3, 4, 6, 7, 8, 9, 14, 15], ou/et sur des techniques exploitant la structure du problème Max-CSP [5, 12].
Dans cet article, nous présentons une technique d'inférence d'inconsistances pour améliorer les bornes inférieures classiques basées sur l'arc consistance. Ceci est effectué en inférant des inconsistances non détéctées par les bornes inférieures classiques. Dans cette technique, pour une valeur d'une variable non instanciée (variable future), nous comptons le nombre d'inconsistances qu'elle a avec les valeurs des variables non instanciées participant dans le calcul de la borne inférieure. Par exemple, dans le cas d'une borne inférieure utilisant les compteurs d'inconsistances et d'inconsistances dirigées (*ic+dac*), pour une valeur a d'une variable non instanciée, nous calculons le nombre d'inconsistances, non calculées par la borne inférieure, entre a et les valeurs des variables futures ayant le minimum *ic+dac*. Par comparaison aux bornes inférieures classiques, notre technique peut rajouter des inconsistances au compteurs *dac* de a même si cette dernière possède des supports sur les variables futures. Nous montrons comment utiliser cette nouvelle borne dans un algorithme de réolution complet pour anticiper les retours arrière et filtrer les valeurs des variables futures. Cette technique a été combinée avec l'algorithme PFC-RDAC [8], le nouvel algorithme PFC-VRDAC (*Partial Forward Checking Value inferring Reversible Directed Arc Consistency*) a été testé sur quelques instances Max-CSP aléatoires. Les premiers résultas sont comparés avec ceux obtenus par PFC-MRDAC [7], un algorithme longtemps considéré comme une référence pour la résolution du Max-CSP. Les gains en temps et en nombre de noeuds visitées, montrent l'efficacité de l'utilisation de l'inférence d'inconsistances avec les bornes classiques pour la résolution du Max-CSP.

Ce article est organisé comme suit. La section 2 présente quelques notions de base relatives au problème CSP. La section 3 décrit l'utilisation de l'inférence d'inconsistances dans le but d'améliorer les bornes inférieures basées sur les compteurs *ic+dac*. Les avantages de la nouvelle borne inférieure par apport aux bornes classiques sont décrits dans la section 4. La procédure de calcul d'inférence d'inconsistances est donnée dans la section 5. La section 6 traite des questions relatives à la mise en ouvre de l'infèrence d'inconsistances dans un branch and bound pour le Max-CSP. La section 7 présente quelques résultas expérimentaux préliminaires obtenus sur les Max-CSP aléatoires. Enfin, la section 8 trace quelques perspectives pour la poursuite de ce travail.

2 Préliminaires

2.1 CSP et Max-CSP

Un problème de satisfaction de contraintes (CSP) est défini par un ensemble fini de variables $X = \{1, \ldots, n\}$, un ensemble de domaines finis et discrêts $D = \{d_1, \ldots, d_n\}$ et un ensemble de contraintes $C = \{C_1, \ldots, C_m\}$. Chaque variable i prend ses valeurs dans le domaine d_i. Chaque contrainte C_j est définie sur un sous ensemble de X appelé sa portée et noté $scope(C_j)$. Une contrainte C_j est un sous ensemble du produit cartésien $d_{i_1} \times d_{i_2} \times \ldots \times d_{i_p}$, avec $\{i_1, \ldots, i_p\} \in scope(C_j)$, elle contient l'ensemble de n-uplets autorisés pour les variables de $scope(C_j)$.
L'instanciation d'une variable i consiste à lui affecter une valeur depuis son domaine d_i. Une instanciation est dite partielle si elle porte sur un sous ensemble de variables de X. Elle est dite complète si elle porte sur toutes les variables de X.

La solution d'un CSP est une instanciation complète qui satisfait toutes les contraintes de l'ensemble C. Si une telle instanciation n'existe pas, le CSP est dit *inconsistant*. Dans ce cas, il est intéressant de chercher l'instanciation complète satisfaisant le plus de contraintes, c'est la satisfaction maximale de contraintes (Max-CSP) [4].

Ces dernières années, le modèle CSP a été étendu vers un modèle plus général le *problème de satisfaction de contraintes valué* (VCSP) [13]. Ce nouveau modèle prend en compte les contraintes *molles* et rend possible le choix entre solutions en associant une *valuation* à chaque n-uplet d'une contrainte. En VCSP, le coût globale d'une instanciation complète est l'agrégation des valuations des n-uplet appartenant à cette instanciation. La solution d'un VCSP est une instanciation complète de coût minimal. Un cas particulier du VCSP, le CSP binaire *pondéré* (WCSP), est défini dans [6]. Dans ce modèle, les contraintes sont soit unaires (C_i), soit binaires (C_{ij}). A chaque valeur a d'une variable i est associé un coût unitaire $C_i(a)$, et à chaque couple $(a, b) \in d_i \times d_j$ d'une contrainte binaire $C_{ij} \in C$ est associé un coût $C_{ij}(a, b)$. Ces coûts sont

des naturels et l'opérateur d'agrégation est l'addition (+). Remarquons que le Max-CSP peut être écrit sous forme d'un WCSP binaire où à tout couple interdit (incohérent) on associe un coût 1 et à un couple autorisé un coût nul. La solution est une instanciation complète minimisant la somme des coûts des couples. Ceci revient à trouver l'instanciation complète minimisant le nombre de contraintes violées.

Dans cet article, $i, j, k, \ldots$ désignentdes variables et $a, b, c, \ldots$ des valeurs. (i, a) désigne la valeur a de la variable i. P désigneune instanciation partielle et F l'ensemble des variables non encore instanciées (libres ou futures). On appel *valeur future* toute valeur d'une variable de F. La valeur (i, a) est inconsistante avec la variable j si elle est incohérente avec toutes les valeurs de d_j. Soit la valeur future (i, a), ic_{ia} est le nombre de valeurs de P inconsistantes avec (i, a). Soit la fonction $dac(i, a, j)$ définie comme suit. $dac(i, a, j)=1$ si (i, a) est inconsistante avec j et si les inconsistances sont comptés de j vers i; 0 sinon. Le nombre d'inconsistances d'arc dirigées, dac_{ia}, est le nombre de variables $j \in F$ tel que $dac(i, a, j)=1$. Soit le Max-CSP $\mathcal{P}$, $\mathcal{P}_P$ est le Max-CSP $\mathcal{P}$ obtenu après propagation de l'instanciation partielle P et $V(\mathcal{P})$ le nombre minimal de contraintes violées dans $\mathcal{P}$.

2.2 Survol des bornes inférieures en Max-CSP

L'un des premiers algorithmes pour la résolution du Max-CSP, PFC [4] utilise une borne inférieure simple. Elle est caluclée en rajoutant à la distance (le coût de l'instanciation partielle P) la somme des ic minimums des variables futures F $(LB(P)_{PFC} = distance(P) + \sum_{i \in F} min_{a \in d_i}(ic_{ia}))$. Cette borne a été améliorée, dans PFC-DAC [14], en considérant les inconsistances entre les variables futures. En définissant un ordre sur les variables et en dirigeant le graphe de contraintes suivant cet ordre, PFC-DAC utilise les compteurs ic et dac et calcule sa borne inférieure comme suit : $LB(P)_{DAC} = distance(P) + \sum_{i \in F} min_{a \in d_i}\{ic_{ia} + dac_{ia}\}$. Une sensible amélioration de PFC-DAC, l'algorithme PFC-RDAC, a été proposée dans [8]. A chaque instanciation d'une variable, PFC-RDAC recalcule une *bonne* orientation des contraintes entre les variables futures dans le but d'augmenter la somme des $ic+dac$ minimums. Dans [7], la propriété d'*arc consistance dirigée* est maintenue durant la résolution. Ce maintien conduit généralement à l'incrémentation des compteurs *dacs* et donc à une possible amélioration de la borne inférieure. Dans *cascaded dac* [15], les compteurs *dacs* sont augmentés en inférant des nouvelles inconsistances en se basant sur l'ordre utilisé dans le calcul des *dacs*. Ce processus d'inférence est limité par l'ordre statique des variables. De plus, il nécessite une gestion des inconsistances redondantes afin de maintenir une borne inférieure valide. Dans [1] la *la consistence d'arc pondérée* (WAC), une borne inférieure ne nécessitant aucun ordre sur les variables est proposée. Contrairement à DAC et RDAC, dans WAC les inconsistances sont calculées dans les deux sens. Afin d'éviter les redondances, deux poids complémentaires $w \leq 1$ et $1-w$ sont associés aux deux sens de la contraintes. Dans [5], une borne inférieure pour le Max-CSP est calculée en partitionnant l'ensemble des contraintes en plusieurs sous ensembles appelés *mini-buckets*. Dans [12] (Conflict Sets), des nouvelles contraintes sont rajoutées au Max-CSP dans le but d'identifier des inconsistances non détectées par l'arc consistance dirigée.

Dans [6, 10], de nouvelles consistances locales (NC,NC*,AC,AC*) sont définies pour le WCSP. Théoriquement, ceci généralise les consistances classiques au WCSP et permet de définir des nouvelles bornes inférieures beaucoup plus élégantes. Cependant, les résultats expérimentaux sur les Max-CSP aléatoires ont montré que les algorithmes (PFC-RDAC et PFC-MRDAC) proposés dans [7, 8] restent plus efficaces. Dans [9], en se basant sur les travaux de [2], des nouvelles formes de consistance locale (*l'arc consistance dirigée*) DAC* and (*l'arc consistance complète dirigée*) FDAC*, sont définies pour le WCSP. Comme dans le cas du Max-CSP, DAC* et FDAC* nécessitent qu'un ordre statique soit défini sur les variables du WCSP. Les algorithmes MDAC* et MFDAC*, qui maintiennent respectivement les consistances DAC* et FDAC*, ont été développés et testés sur des Max-CSPs aléatoires. Les résultats obtenus ont montré que le maintien de FDAC* semble être le meilleure niveau de consistance pour la résolution des Max-CSP aléatoires. Récemment dans [3], une nouvelle forme d'arc consistance est définie, c'est *la conhérence d'arc existentielle* EDAC*. EDAC* renforce la propriété de FDAC* afin de se rapprocher le plus de la consistance d'arc complète. MEDAC*, un algorithme qui maintient la propriété EDAC*, a été developpé et testé sur des WCSPs aléatoires et réels. Les tests ont montré que MEDAC* est plus efficace que MFDAC* sur de nombreuses classes de problèmes.

3 Amélioration des bornes inférieures classiques par inférence d'inconsistances

Sans perte de généralité, nous ne considérons que les bornes inférieures basées sur les compteurs $ic+dac$. L'inférence d'inconsistances peut également être utilisée avec les bornes inférieures basées sur l'arc consistance comme l'*arc consistance podérée* WAC.

3.1 L'inférence d'inconsistances après l'instanciation d'une variable

Pendant l'instanciation d'une variable i par la valeur $a \in d_i$, le branch and bound calcule une borne inférieure du nombre de contraintes violées par l'instanciation partielle $P \cup \{(i,a)\}$. Cette borne inférieure est donnée par la formule suivante, $LB(P,i,a) = distance(P) + ic_{ia} + dac_{ia} + \sum_{j \in F} min_{a \in d_j}\{ic_{ja} + dac_{ja}\}$. La somme $ic_{ia} + dac_{ia}$ représente la contribution de la valeur (i,a) dans cette borne. Sous certaines conditions, cette contribution peut être augmentée même si (i,a) est consistante avec quelques variables futures F. De plus, à cause de l'orientation des contraintes, les inconsistances avec quelques variables de F ne sont pas comptées même si (i,a) est inconsistante avec ces variables. En utilisant ces remarques, nous proposons dans ce qui suit une technique d'inférence d'inconsistances afin d'améliorer les bornes inférieures basées sur *ic+dac*.
Soit (i,a) la valeur actuellement instanciée à i et soit $j \in F$ une variable future partageant une contrainte avec i. Considérons la situation suivante :

- b est l'unique valeur de j minimisant la somme *ic+dac*.
- le couple (i,a) et (j,b) est incohérent et $dac(i,a,j) = dac(j,b,i) = 0$.

Soit $k \geq 0$ le nombre de variables futures vérifiant les conditions ci-dessus et partageant une contrainte avec i. Alors la borne inférieure associée à la valeur (i,a), $LB(P,i,a)$, peut être incrémentée de k. Comme les inconsistances rajoutées ne sont calculées ni par *dacs*, ni par *ics*, $LB(P,i,a)+k$ peut être considérée comme une borne inférieure du Max-CSP.

Dans le précédent paragraphe, nous avons considéré les variables futures avec un minimum unique. L'inférence d'inconsistances est aussi applicable avec les variables futures possédant plusieurs valeurs atteignant le minimum *ic+dac*. Soit $\mathcal{M}(j)$ l'ensemble des valeurs de j ayant une somme *ic+dac* minimale. La quantité à rajouter à la contribution de la valeur (i,a) dans $LB(P,i,a)$ est donnée par : $Infer(i,a) = \sum_{j \in F} min_{b \in \mathcal{M}(j)}\{r(i,a,j,b)\}$, où

$$r(i,a,j,b) = \begin{cases} 1 & \text{si } (i,a) \text{ et } (j,b) \text{ sont incohérents} \\ & \text{et, } dac(i,a,j) = 0 \text{ et } dac(j,b,i) = 0 \\ 0 & \text{sinon} \end{cases}$$

La propriété suivante montre que $LB_{Infer}(P,i,a) = LB(P,i,a) + Infer(i,a)$ est une borne inférieure du nombre de contraintes violées par toute instanciation complète extension de l'instanciation partielle $P \cup \{(i,a)\}$.

Propriété 1. *Soient le Max-CSP $\mathcal{P}$ et l'instanciation partielle P, $LB_{Infer}(P,i,a)$ est une borne inférieure de $V(\mathcal{P}_{P \cup \{(i,a)\}})$.*

Preuve. Soit l'instanciation complète $A = (a_1, a_2, \ldots, a_n)$ extension de P avec a affectée à i. Le coût de A, le nombre de contraintes violées par A, est noté par $distance(A)$. Soit $LB(A,i,a) = distance(P) + ic_{ia} + dac_{ia} + \sum_{j \in F}(ic_{ja_j} + dac_{ja_j})$ où :

- $distance(P)$ repésente le nombre de contraintes violées par l'instanciation partielle P.
- $ic_{ia} + dac_{ia}$ représente le nombre de contraintes violées entre (i,a) et l'instanciation partielle P et une sous estimation du nombre de contraintes violées entre (i,a) et les variables futures.
- $\sum_{j \in F}(ic_{ja_j} + dac_{ja_j})$ est une sous estimation du nombre de contraintes violées entre les valeurs $a_j, j \in F$ de A, et le nombre de contraintes violées entre les valeurs $a_j, j \in F$ de A et l'instanciation partielle $P \cup \{(i,a)\}$.

Par définition nous avons $distance(A) \geq LB(A,i,a)$. Nous allons d'abord prouver que $ic_{ja_j} + dac_{ja_j} + r(i,a,j,a_j) \geq min_{a \in d_j}\{dac_{ja} + ic_{ja}\} + min_{b \in \mathcal{M}(j)}\{r(i,a,j,b)\}$. Comme $ic_{ja_j} + dac_{ja_j} \geq min_{a \in d_j}\{dac_{ja} + ic_{ja}\}$ alors l'inégalité est vérifiée si $r(i,a,j,a_j) \geq min_{b \in \mathcal{M}(j)}\{r(i,a,j,b)\}$.
Considérons maintenant le cas où $r(i,a,j,a_j) = 0$ et $min_{b \in \mathcal{M}(j)}\{r(i,a,j,b)\} = 1$ et supposons que $ic_{ja_j} + dac_{ja_j} + r(i,a,j,a_j) < min_{a \in d_j}\{dac_{ja} + ic_{ja}\} + min_{b \in \mathcal{M}(j)}\{r(i,a,j,b)\}$, alors $ic_{ja_j} + dac_{ja_j} < min_{a \in d_j}\{dac_{ja} + ic_{ja}\} + 1$ ce qui implique que $ic_{ja_j} + dac_{ja_j} \leq min_{a \in d_j}\{dac_{ja} + ic_{ja}\}$. Donc $a_j \in \mathcal{M}(j)$ et $r(i,a,j,a_j) = 1$, ce qui est en contradiction avec les hypothèses.
En sommant sur l'ensemble des variables futures, nous avons $\sum_{j \in F}(ic_{ja_j} + dac_{ja_j} + r(i,a,j,a_j)) \geq \sum_{j \in F} min_{a \in d_j}\{dac_{ja} + ic_{ja}\} + \sum_{j \in F} min_{b \in \mathcal{M}(j)}\{r(i,a,j,b)\}$. Cette inégalité peut être écrite comme suit. $\sum_{j \in F}(ic_{ja_j} + dac_{ja_j} + r(i,a,j,a_j)) \geq \sum_{j \in F} min_{a \in d_j}\{dac_{ja} + ic_{ja}\} + Infer(i,a)$. En rajoutant $distance(P) + ic_{ia} + dac_{ia}$ aux deux côtés de l'inégalité nous avons $LB(A,i,a) \geq LB_{Infer}(P,i,a)$. □

La nouvelle borne inférieure $LB_{Infer}(P,i,a)$ est au moins égale à $LB(P,i,a)$ et au plus égale à $LB(P,i,a) + \rho(i)$, où $\rho(i)$ est le nombre de variables partageant une contraintes avec i.

3.2 L'inférence d'inconsistances pendant le filtrage

Le processus de filtrage est utilisé par tous les solveurs Max-CSP, avant et au cours de la recherche. Le

filtrage est utilisé pour supprimer les valeurs qui n'appartiendront pas à la solution du Max-CSP. Son but est de rendre le problème plus facile à résoudre. La borne inférieure basée sur l'ic+dac, $LB_{Filt}(P,i,a)$ utilisée pour le filtrage est donnée par la formule suivante, $distance(P)$+ic_{ia}+dac_{ia}+$\sum_{j\in F-\{i\}} min_{a\in d_j}\{ic_{ja}+dac_{ja}\}$.
En plus de son utilisation avec $LB(P,i,a)$, l'inférence d'inconsistances peut être utilisée pour augmenter la valeur de $LB_{Filt}(P,i,a)$.
Soit (i,a) une valeur future et soit $j \neq i$ une variable future partageant une contrainte avec i et vérifiant les conditions suivantes :

- j possède une seule valeur b minimisant la somme ic+dac. La valeur b est incohérente avec (i,a),
- $dac(i,a,j) = dac(j,b,i) = 0$.

S'il y a $k \geq 0$ variables futures de $F-\{i\}$ vérifiant les conditions ci-dessus, alors $LB_{Filtr}(P,i,a)$ peut être incrémentée de k. Cette somme, qu'on notera $LB_{InferFilt}(P,i,a)$ est exprimée comme suit :

$$LB_{InferFilt}(P,i,a) = LB_{Filtr}(P,i,a)+InferFilt(i,a)$$

avec

$$InferFilt(i,a) = \sum_{j\in F-\{i\}} min_{b\in\mathcal{M}(j)}\{r(i,a,j,b)\}$$

Propriété 2. *Soit $\mathcal{P}_{P\cup\{(i,a)\}}$ le Max-CSP obtenu après la propagation de P, alors $LB_{InferFilt}(P,i,a)$ est une borne inférieure de $V(\mathcal{P}_{P\cup\{(i,a)\}})$.*

Preuve. La preuve est semblable à celle de la propriété 1. □

La valeur future (i,a) est supprimée du domaine d_j si $LB_{InferFilt}(P,i,a) \geq ub$, où ub est la borne supérieure de $\mathcal{P}$.

4 Discussion

Le calcul de $LB_{Infer}(P,i,a)$ est une anticipation de la propagation de l'instanciation partielle $P\cup\{(i,a)\}$ sur les domaines des variables futures. Cependant, ce calcul est effectué sans mise à jour des compteurs ic et sans suppression de valeurs futures.
Comme $LB_{InferFiltr}(P,i,a)$ est au moins égale à $LB_{Filtr}(P,i,a)$, elle supprimera plus de valeurs et l'arbre de recherche résultant est plus réduit (voir la Figure 1). La figure 1 montre les arbres de recherche développés avec et sans inférence d'inconsistances pendant la résolution du Max-CSP aléatoire $\langle 4,3,7/9,1\rangle$ (voir la section 7 pour plus de détails concernant les CSPs aléatoires). Le branch and bound sans inférence d'inconsistances développe 5 noeuds de plus que celui

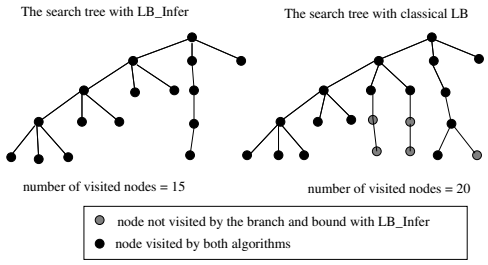

Figure 1: Arbres de recherche développés avec et sans LB_{Infer} sur le Max-CSP aléatoire $\langle 4,3,7/9,1\rangle$.

utilisant LB_{Infer} et $LB_{InferFiltr}$. Les expérimentations de la section 7 confirme l'efficacité de LB_{Infer} et plus particulièrement de $LB_{InferFilt}$ dans le filtrage des domaines des variables futures sur une variété de problèmes aléatoires.

4.1 LB_{Infer}, DAC* et FDAC*

Les consistances locales DAC* et FDAC* transforment le réseau de contraintes initial en un réseau équivalent vérifiant les propriétés DAC* et FDAC*. Ceci est effectué en utilisant les procédures Extend() et Project() définies dans [6].
La figure 2 montre un exemple d'application de FDAC* sur une contrainte c_{ij} avec $i<j$. Dans cette figure, les coûts des contraintes unaires sont représentés dans des petits cercles et les coûts binaires sont représentés sur les arcs connectant chaque couple de valeurs. L'absence d'un arc entre deux valeurs signifie que le coût binaire entre ces deux valeurs est nul. Nous supposons dans cet exemple que la borne inférieure $lb=0$ et que la borne supérieure $ub=2$.
Dans la figure 2, la valeur (i,a) ne possède pas de support complet[1] sur la variable j. En appliquant FDAC* pour cette valeur (suivant l'ordre $i<j$) le coût $C_i(a)$ est incrémenté d'une unité. L'application de FDAC* correspond exactement au résultat de l'application de l'inférence d'inconsistances pour la valeur (i,a). Notons que la valeur (i,a) sera supprimée car lb+$C_i(a) \geq ub$. Le calcul de $LB_{infer}(P,i,a)$ sur la contrainte C_{ij} détectera cette inconsistance et supprimera la valeur (i,a). De plus, comme le calcul de LB_{infer} ne dépend pas de l'orientation de la contrainte, elle est appliqué dans l'autre sens et rajoute une inconsistance au coût de la valeur (j,a), cette inconsistance est ignorée par FDAC* à cause de l'orientation de la contrainte.
L'avantage d'utiliser LB_{Infer} et $LB_{InferFiltr}$ est qu'on n'a besoin de maintenir aucune forme de consistance locale durant la recherche. De plus, aucun or-

[1] on dit que $b\in d_j$ est un support complet de $a\in d_i$ sur la contraintes C_{ij} si $i<j$ et $C_{ij}(a,b)+C_i(b)=0$

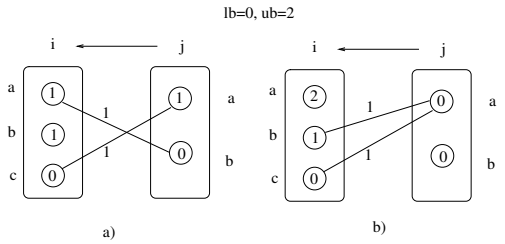

Figure 2: Application de FDAC* sur C_{ij}. (a) la valeur (i, a) n'est pas DAC. (b) Après extension et projection des coûts, (i, a) possède maintenant un support complet sur C_{ij}.

dre sur les variables n'est nécessaire pour l'application de LB_{Infer}. Le but principal derrière LB_{Infer} et $LB_{InferFiltr}$ est d'augmenter la borne inférieure afin d'avoir un filtrage plus robuste. Dans FDAC* et DAC*, le but est de maintenir la propriété FDAC* durant la recherche. Ceci est effectué en utilisant la projection et l'extension des coûts des contraintes binaires et unaires et la modification du réseau de contraintes. Dans FDAC* et DAC*, une ordre statique sur les variables du Max-CSP est indispensable alors que LB_{Infer} et $LB_{InferFiltr}$ n'ont pas cette limitation.

4.2 Vers une généralisation au WCSP

Comme montré précedemment (figure 2), l'application de FDAC* sur une contrainte dans un sens est équivalent à l'application de LB_{Infer} suivant le même sens. Trouver un support complet pour une valeur (i, a) sur la contrainte C_{ij} revient à chercher une valeur $c \in d_j$ tel que $c = argmin_{b \in d_j}\{C_{ij}(a, b) + C_j(b)\}$. Après extension et projection du coût $C_{ij}(a, c) + C_j(c)$ vers $C_i(a)$, la valeur (j, c) devient un support complet sur C_{ij} pour (i, a). Ce processus est similaire à celui utilisé par LB_{Infer}. Ainsi, l'application de LB_{Infer} peut être généralisée au problème WCSP. Dans le WCSP, l'inférence d'inconsistances pour une valeur (i, a) sur la contrainte C_{ij} consiste à rajouter le coût $C_{ij}(a, c) + C_j(c)$ à la borne inférieure globale. Mais, FDAC* nécessite un ordre sur les variables afin de maintenir la propriété d'arc consistance complète dirigée alors que LB_{Infer} n'a pas cette limitation. Pour le filtrage, FDAC* supprime une valeur (i, a) si $(lb + C_i(a)) \geq ub$. Avec LB_{Infer}, (i, a) est supprimée si $(distance(P) + C_i(a) + (\sum_{j \in F} min_{b \in d_j}\{C_{ij}(a, b) + C_j(b)\})) \geq ub$.

5 Calcul des inconsistances inférées en Max-CSP

Comme le nombre de variables futures partageant une contrainte avec i est au plus égale à n et que pour chaque variable future j, nous testons la consistance de (i, a) avec au plus d valeurs, alors la complexité de LB_{Infer} ($LB_{InferFilt}$) est en $O(nd)$, où d est la taille du plus grand domaine et n le nombre de variables.
Le calcul de $LB_{Infer}(P, i, a)$ et $LB_{InferFiltr}(P, i, a)$ en considérant tous les minimums d'une variable future fournit une borne inférieure de qualité mais s'avère coûteux en pratique. Ainsi, nous préférons calculer LB_{Infer} en ne considérant que les variables futures avec un minimum unique (voir **Algorithm 1**). Ceci conduit à un bon compromis entre la qualité de la borne inférieure et l'effort nécessaire pour la calculer. L'algorithme 1 montre le calcul et l'utilisation de l'inférence d'inconsistances dans le filtrage des valeurs futures. P représente l'instanciation partielle courante, D l'ensemble des domaines des variables futures et G le graphe de contraintes. La fonction SingleMin(j) est *vraie* s'il y a un seul minimum dans D_j. La procédure Prune(j, b) supprime la valeur (j, b) du domaine D'_j. Pour chaque valeur future (j, b), on regarde d'abord si on peut la supprimer sans inférer d'inconsistances en utilisant les bornes inférieures classiques. Sinon, on calcule $LB_{InferFiltr}(j, b)$ (la boucle de la ligne 2) et on regarde si on peut maintenant la supprimer (ligne 3).

Algorithm 1: PruneValues(P, F, D, G)

```
begin
    for j ∈ F do
        for b ∈ D_j do
1           if (LB(P, j, b) ≥ ub) then
                Prune(j, b);
            else
                cont ← 0;
2               for k ∈ F connected to j and k ≠ j do
                    if (SingleMin(k)) then
                        e ← argmin_{c∈D_k}{ic_kc + dac_kc};
                        cont ← cont + r(j, b, k, e);
3               if (LB(P, j, b) + cont ≥ ub) then
                    Prune(j, b);
    return new domains;
end
```

6 L'algorithme PFC-VRDAC

Nous avons implémenté LB_{Infer} et $LB_{InferFiltr}$ dans l'algorithme PFC-RDAC. Rappelons que PFC-RDAC est un algorithme de type branch and bound pour la résolution du Max-CSP. Après chaque instanciation d'une variable, PFC-RDAC réoriente le graphe de contraintes des variables futures afin d'augmenter

la valeur de la borne inférieure.
L'algorithme résultant de la combinaison de RDAC et de l'inférence d'inconsistances est appelé PFC-VRDAC. L'idée de PFC-VRDAC est d'utiliser LB_{Infer} avant chaque propagation d'une instanciation et $LB_{InferFilt}$ durant le filtrage afin de supprimer les valeurs non filtrées par les bornes inférieures *ic+dac*.
Dans PFC-VRDAC (**Algorithm 2**), à chaque noeud (instanciation de de i par a), on compte d'abord son $LB_{Infer}(P, i, a)$ (ligne 1). Si $LB_{Infer}(P, i, a) \geq ub$ un retour arrière est effectué. Sinon, l'instanciation partielle courante est propagé sur les domaines des variables futures (La fonction LookAhead ligne 2). Si durant la propagation un des domaines futures devient vide, un retour arrière est effectué (voir la ligne 3) puisque la fonction continue() retourne *faux* si un des domaines est vide. Sinon, comme dans PFC-RDAC, PFC-VRDAC réoriente les contraintes du graphe des variables futures à l'aide de l'heuristique GreedyOpt() (ligne 4) dans le but d'augmenter la valeur de la borne inférieure. Si après GreedyOpt() aucun échec n'est détecté, PFC-VRDAC appel la fonction PruneValues (ligne 5) en renforçant le filtrage par l'utilisation des inconsistances inférées. Si aucun domaine n'est vidé aprés l'appel de PruneValues, alors un nouvel appel à PFC-VRDAC est effectué. Quand toutes les variables sont instanciées ($F = \phi$), PFC-VRDAC garde une trace de la meilleure instanciation $bestS$ ainsi que son coût et met à jour la borne supérieure du problème ub.

Algorithm 2: PFC-VRDAC(P, d, F, D, G^F)

```
   begin
       if (F = φ) then
           ub ← d;
           bestS ← P;
       else
           i ← Pop (F);
           for a ∈ D_i do
               d' ← d + ic_ia + dac_ia;
1              if (LB_Infer(P, i, a) < ub) then
2                  D' ← LookAhead(d', i, a, F, D, G^F);
3                  if (continue(D')) then
4                      nG^F ← GreedyOpt(G^F, F, D');
                       if (continue(D')) then
5                          D' ←
                           PruneValues(P∪{a_i}, F, D', nG^F);
                           if (continue(D')) then
                               PFC-VRDAC(P∪
                               {a_i}, d', F, D', nG^F);
   end
```

7 Expérimentations

Cette section évalue les performances de PFC-VRDAC, en terme de temps et de nombre de noeuds visités, sur des Max-CSP aléatoires. On présente la version qui considère les variables futures avec un minimum unique. La version la plus générale, dans laquelle l'inférence d'inconsistances est effectuée en considérant tous les minimums, a été implémentée et testée. Malgré sa capacité à filtrer considérablement les valeurs futures, elle s'est avérée inefficace en terme de temps de résolution.
Des algorithmes récents qui maintiennent les propriétés AC et AC* pour le WCSP [10] ont été testés sur des Max-CSP aléatoires. Les résultats obtenus ont été comparés avec ceux obtenus par PFC-RDAC. Cette comparaison a montré que ces algorithmes sont moins performants que PFC-RDAC sur les problèmes aéatoires.
Une classe de CSP binaire aléatoire est caractérisée par le quadruplet $\langle n, d, p_1, p2\rangle$, où n est le nombre de variables, d la taille de chaque domaine, p_1 la connectivité du graphe de contraintes définie par le rapport du nombre de contraintes existantes et le nombre de contraintes possibles $\frac{n(n-1)}{2}$, p_2 est la dureté de chaque contrainte, elle est donnée par le rapport du nombre de couples interdits et le nombre totale de couples possibles $d \times d$. Les contraintes entre variables et les couples interdits dans chaque contrainte sont choisis aléatoirement [11]. En utilisant ce modèle, nous avons testé PFC-VRDAC sur les mêmes classes utilisées dans [8] :

(1) $\langle 10, 10, 1, p_2\rangle$	(2) $\langle 15, 5, 1, p_2\rangle$
(3) $\langle 15, 10, 50/105, p_2\rangle$	(4) $\langle 20, 5, 100/190, p_2\rangle$
(5) $\langle 25, 10, 37/300, p_2\rangle$	(6) $\langle 40, 5, 55/780, p_2\rangle$

Les classes (1) et (2) représentent des Max-CSP fortement connectés. Les classes (3) et (4) représentent des Max-CSP moyennement connectés et les classes (5) et (6) représentent des Max-CSP faiblement connectés. Pour chaque classe de problème et pour chaque valeur du paramètre $p2$, 50 instances sont générées. Chaque instance est respectivement résolue par PFC-MRDAC et PFC-VRDAC. L'heuristique de sélection de variables pour les deux algorithmes est *taille du domaine/degré futur*. Le domaine d'une variable est trié suivant l'ordre croissant des compteurs *ic+dac*. Les expérimentations ont été effectuées sur une machine `i386` avec `1028264 kB` de mémoire RAM et un processeur de `2.66Ghz`. La borne supérieure initiale est initialisé à la valeur maximale qu'un entier en C peut contenir.
Les **Figure. 3**, **Figure. 4** et **Figure. 5** donnent respectivement le nombre moyen de noeuds visités par PFC-MRDAC et PFC-VRDAC, sur les problèmes fortement, moyennement et faiblement connectés. Pour toutes les classes, PFC-VRDAC visite moins de noeuds que PFC-MRDAC. Le gain s'accroît avec la dureté de la contrainte et atteint son maximum aux alentours du point de transition. En moyenne, PFC-VRDAC visite 47% de noeuds en moins que PFC-MRDAC. Pour les problèmes complets (**Figure 3**) et

dans la région difficile, PFC-VRDAC visite deux fois moins de noeuds que PFC-MRDAC. On a le même comportement pour les problèmes moyennement connectés (**Figure 4**) et les problèmes faiblement connectés (**Figure 5**). Ces résultats confirment que $LB_{InferFilt}$ filtre plus de valeurs que les bornes inférieures classiques. Les **Figure. 6**, **Figure. 7** et **Figure. 8** présentent la moyenne du temps (en secondes) qu'ont mis les deux algorithmes pour résoudre à l'optimalité les problèmes des six classes. Les résultats obtenus concordent avec ceux des noeuds visités. Pour tous les problèmes, PFC-VRDAC met moins de temps pour prouver l'optimalité que PFC-MRDAC. En moyenne, PFC-VRDAC est 25% plus rapide que PFC-MRDAC. Le gain s'accroît avec la dureté de la contrainte et atteint son maximum aux alentours du point de transition.

8 Conclusion

Il est bien connu que la borne inférieure utilisée dans un algorithme de branch and bound doit être la plus proche possible de la valeur du problème et pas trop coûteuse à calculer. Utilisant ce principe, nous avons présenté une nouvelle borne inférieure LB_{Infer} ($LB_{InferFilt}$) basée sur les bornes inférieures classiques et utilisant une nouvelle technique d'inférence. Cette nouvelle borne a été utilisée dans un algorithme de branch and bound qui a été testé sur une variété de problèmes Max-CSP aléatoires. Les résultats obtenus ont montré des gains significatifs en terme de nombre de noeuds visités et en temps CPU. La nouvelle borne inférieure LB_{Infer} ($LB_{InferFilt}$) semble être un bon compris entre qualité de la borne inférieure et le coût nécessaire pour son calcul. Tous ces aspects nous encouragent à poursuivre ce travail et à l'étendre au WCSP (section 4.2). Des expérimentations sont actuellement effectuées sur des problèmes réels.

References

[1] M.S. Affane and H. Bennaceur. A weighted arc consistency technique for max-csp. In *Proceedings of the 13 th ECAI*, pages 209–213, 1998.

[2] M Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systemes*, 134(3), 2003.

[3] S de Givry, F Heras, J Larrosa, and M Zytnicki. Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In *International Join Conference on Artificial Intelligence IJCAI 2005*, 2005.

[4] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(1-3):21–70, 1992.

[5] K. Kask. New search heuristics for max-csp. In *Proceeding of CP'2000*, pages 262–277, 2000.

[6] J. Larrosa. On arc and node consistency in weighted csp. In *Proceedings of AAAI'02*, 2002.

[7] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence*, 107(1):149–163, 1999.

[8] J. Larrosa, P. Meseguer, T. Schiex, and G. Verfaillie. Reversible DAC and other improvements for solving Max-CSP. In *AAAI/IAAI*, pages 347–352, 1998.

[9] J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted csp. In *In Proceedings of the International Joint Conference on Artificial Intelligence IJCAI-03*, 2003.

[10] J. Larrosa and T. Schiex. Solving Weighted CSP by Maintaining Arc Consistency. *Artificial Intelligence*, 159(1-2):1–26, November 2004.

[11] P. Prosser. Binary constraint satisfaction problem: Some are harder than others. In *Proceedings of the ECAI-94*, pages 95–99, 1994.

[12] J.C Regin, T. Petit, C. Bessière, and J F. Puget. New lower bounds of constraint violations for over-constrained problems. In *Proceedings of Principles and Practice of Constraint Programming*, pages 332–345, 2001.

[13] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In Chris Mellish, editor, *Proceedings of the IJCAI*, Montreal, CA, 1995.

[14] Richard Wallace. Directed arc consistency preprocessing as a strategy for maximal constraint satisfaction. In Manfred Meyer, editor, *Proceedings ECAI'94 Workshop on Constraint Processing*, Amsterdam, 1994.

[15] Richard J Wallace. Enhancements of branch and bound methods for the maximal constraint satisfaction proble. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, Portland, Oregon, USA, 1996.

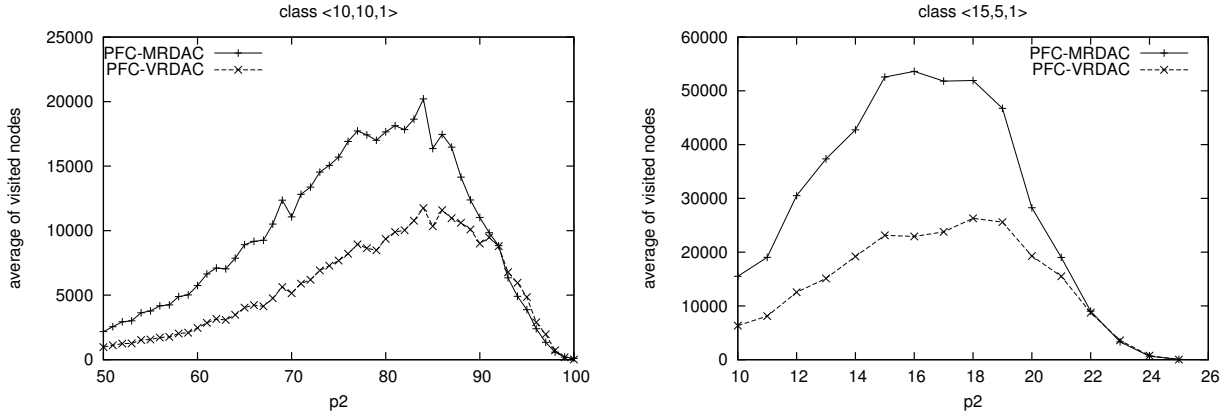

Figure 3: Moyenne du nombre de noeuds visités par PFC-MRDAC et PFC-VRDAC sur des CSP complets

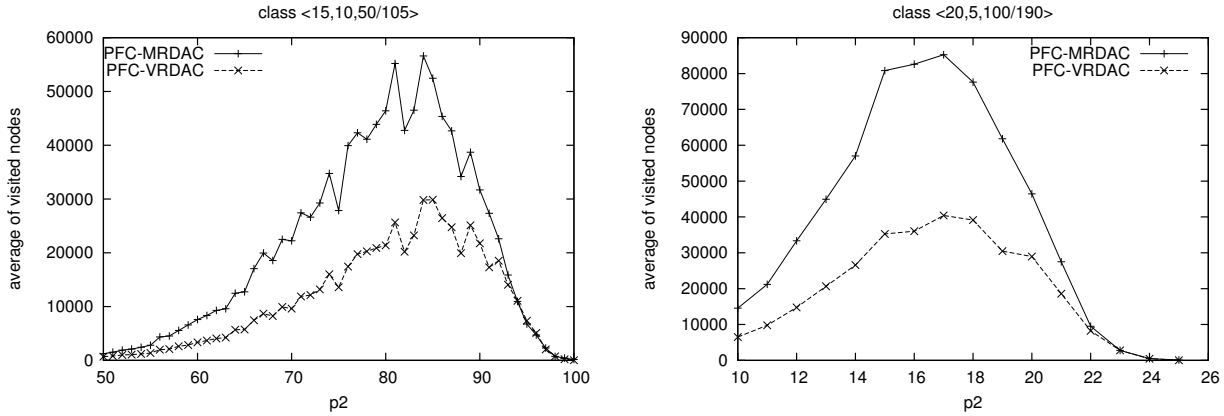

Figure 4: Moyenne du nombre de noeuds visités par PFC-MRDAC et PFC-VRDAC sur des CSP moyennement connectés

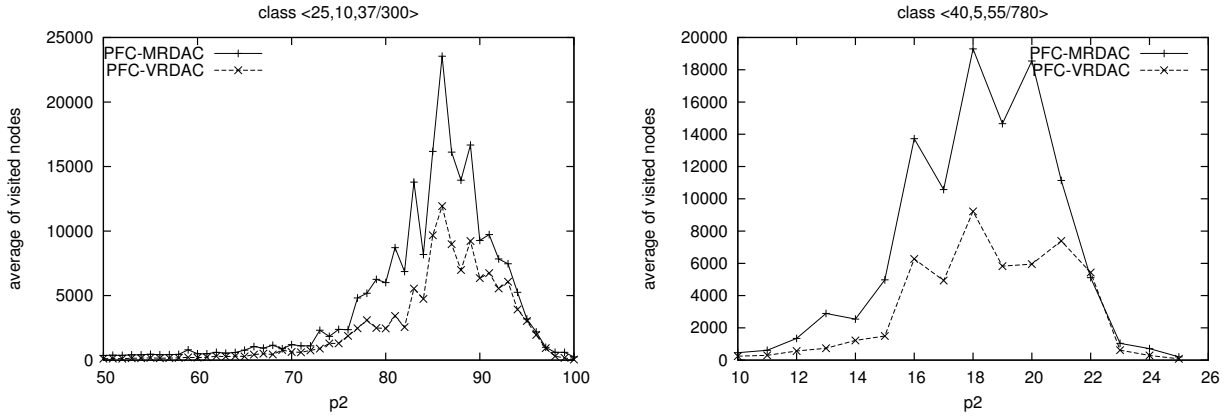

Figure 5: Moyenne du nombre de noeuds visités par PFC-MRDAC et PFC-VRDAC sur des CSP faiblement connectés

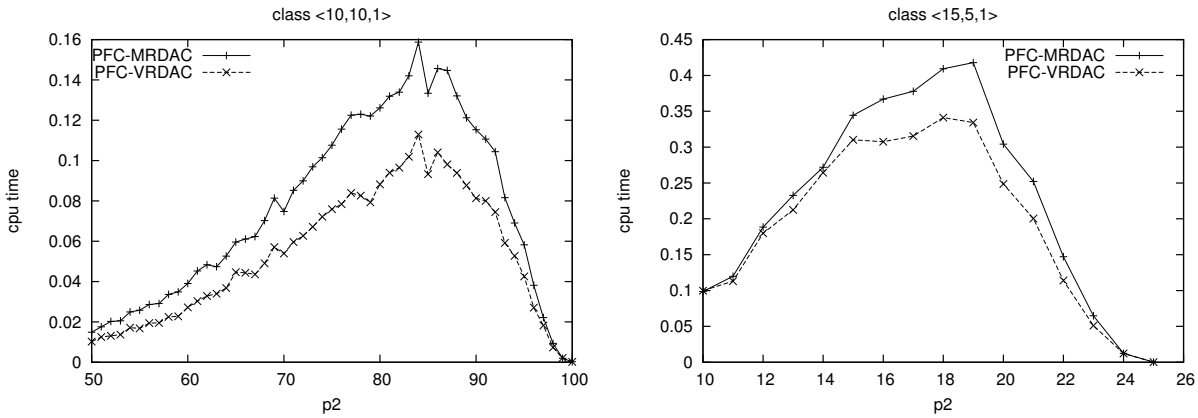

Figure 6: Moyenne du temps CPU (en secondes) de PFC-MRDAC et PFC-VRDAC sur les problèmes CSP complets

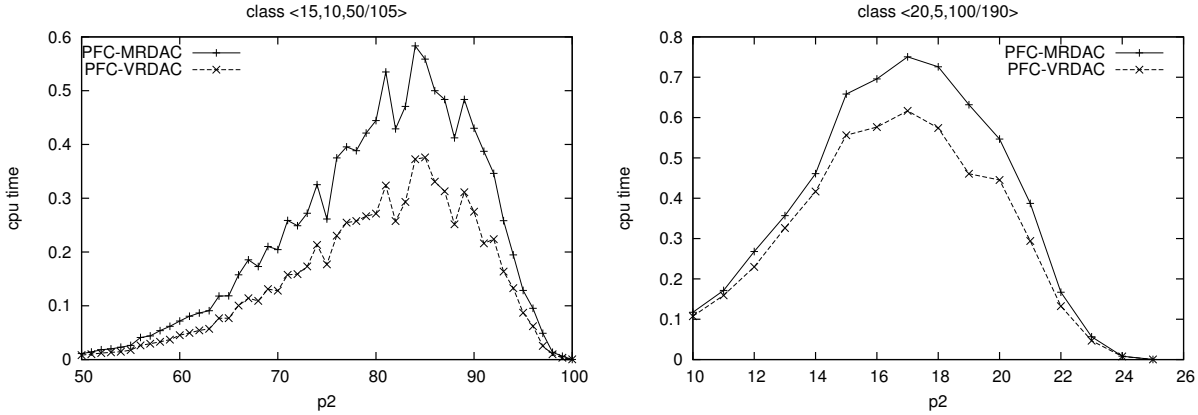

Figure 7: Moyenne du temps CPU (en secondes) de PFC-MRDAC et PFC-VRDAC sur les problèmes CSP moyennement connectés

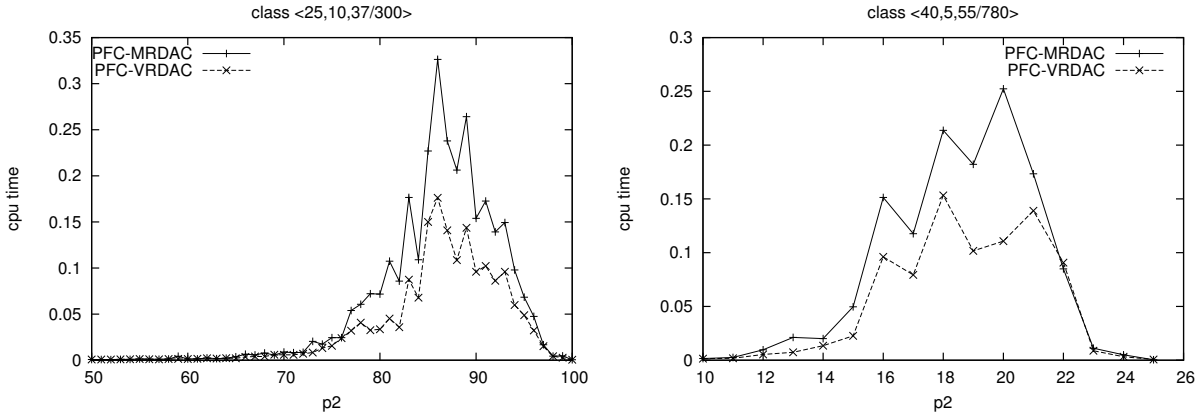

Figure 8: Moyenne du temps CPU (en secondes) de PFC-MRDAC et PFC-VRDAC sur les problèmes CSP faiblement connectés