Cohérences Locales Paramétrées *

Amine Balafrej[1,2] **Christian Bessiere**[1] **Remi Coletta**[1] **El Houssine Bouyakhf**[2]

[1]LIRMM, Université de Montpellier, France
[2]LIMIARF-FSR, Université Mohammed V-Agdal, Maroc
{balafrej,bessiere,coletta}@lirmm.fr bouyakhf@fsr.ac.ma

Résumé

Les solveurs de contraintes maintiennent uniformément le même niveau de cohérence locale (généralement la cohérence d'arc) indépendamment de l'instance du problème à résoudre. Nous proposons le concept de *cohérence locale paramétrée*, une approche originale qui permet d'ajuster le niveau de cohérence locale en fonction de l'instance du problème à résoudre et en fonction de la partie du problème dans laquelle s'effectue la propagation. Nous n'utilisons pas comme paramètre l'une des caractéristiques de l'instance, comme le font les portefeuilles de solveurs. Nous utilisons comme paramètre la *stabilité* des valeurs, une caractéristique basée sur l'état de l'algorithme de cohérence d'arc durant son exécution. Les cohérences locales paramétrées appliquent à une valeur, la cohérence d'arc ou un autre niveau de cohérence plus élevé, selon que la stabilité de cette valeur est en dessus ou en dessous d'un seuil donné. L'approche que nous proposons permet d'obtenir un bon compromis entre la capacité d'une cohérence locale à supprimer des valeurs et le coût en temps pour atteindre ce niveau de cohérence dans un réseau de contraintes. Nous validons notre approche sur différents problèmes de la dernière compétition des solveurs de contraintes.

1 Introduction

Les propriétés de cohérence locale constituent un des points forts de la programmation par contraintes (PPC). En effet, la propagation des contraintes en appliquant une de ces propriétés permet aux solveurs de réduire l'espace de recherche en supprimant les valeurs localement inconsistantes. La cohérence d'arc (AC) [5] est la plus célèbre et la plus ancienne propriété de cohérence locale. Elle a la bonne propriété de supprimer des valeurs localement inconsistantes sans modifier la structure du réseau de contraintes. La cohérence d'arc est aussi le niveau standard de cohérence locale maintenue par les solveurs de contraintes. Plusieurs autres propriétés de cohérence locale, ne modifiant pas la structure du réseau de contraintes, qui sont plus fortes que l'AC ont été proposées (telles que la max-cohérence de chemin restreinte (maxRPC) [4] ou la singleton cohérence d'arc (SAC) [4]). Bien qu'elles soient capables de supprimer plus de valeurs inconsistantes que l'AC, ces cohérences sont rarement utilisées en pratique en raison du coût élevé de leur maintien durant la recherche. Cependant, dans certains problèmes, le maintien de la cohérence d'arc devient un surcoût en raison du grand nombre de révisions inefficaces des contraintes, ce qui est pénalisant en temps CPU. Par exemple, dans [6], Stergiou observe que lors de la résolution de l'instance scen11, du problème d'allocation de fréquences de liaison radio (RLFAP) avec un algorithme qui maintient la cohérence d'arc, seulement 27 parmi les 4102 contraintes du problème étaient identifiées comme causant un domaine vide et 1921 contraintes n'ont supprimé aucune valeur. Le choix de la bonne propriété de cohérence locale pour résoudre efficacement un problème de satisfaction de contraintes consiste à trouver un bon compromis entre la capacité de cette cohérence à supprimer des valeurs inconsistantes, et le coût des algorithmes qui permettent de l'atteindre dans un réseau de contraintes. Stergiou suggère d'exploiter la puissance des cohérences fortes pour réduire l'espace de recherche, tout en évitant le coût élevé de leur maintien dans l'ensemble du réseau. Il propose une approche basée sur des heuristiques, qui exploitent des événements qui peuvent se produire durant la propagation, pour adapter dynamiquement le niveau de cohérence locale (AC ou maxRPC) appliqué aux contraintes individuellement. Cette méthode permet d'élaguer plus que la cohérence d'arc et moins que la max-cohérence de chemin restreinte. Mais l'élagage obtenu n'est pas caracté-

*Ce travail a bénéficié du soutien du projet européen ICON (FP7-284715).

ristique d'une propriété de cohérence locale. Nous pouvons converger vers différentes fermetures selon l'ordre de propagation des contraintes.

Dans ce papier nous définissons la notion de stabilité pour les valeurs. Une notion originale qui n'est pas basée sur les caractéristiques de l'instance à résoudre, mais basée sur l'état de l'algorithme de propagation de la cohérence d'arc pendant son exécution. En se basant sur cette notion, nous proposons les *cohérences paramétrées*, une approche originale qui permet d'ajuster le niveau de cohérence locale utilisé pour résoudre une instance donnée. Une cohérence paramétrée p-LC maintient un niveau intermédiaire de cohérence entre l'AC et une autre cohérence LC plus forte que la cohérence d'arc. Le pouvoir d'élagage de p-LC dépend du paramètre p. Cette approche nous permet de trouver un bon compromis entre la puissance d'élagage d'une cohérence locale et le coût en terme de temps CPU des algorithmes qui permetent de l'atteindre. Nous appliquons le concept de cohérence paramétrée p-LC aux deux propriétés de cohérence locale plus fortes que l'AC les plus connues, qui ne modifient pas la structure du réseau, à savoir maxRPC et SAC. Nous décrivons l'algorithme p-maxRPC3, basé sur maxRPC3 [1], qui réalise la p-max cohérence de chemin restreinte ainsi que l'algorithme p-SAC1, qui réalise la p-singleton cohérence d'arc. Nous évaluons expérimentalement l'intérêt d'utiliser la cohérence locale paramétrée. Et nous montrons qu'en faisant le bon choix du paramètre p, nous profitons du coût réduit de calcul de la cohérence d'arc et l'efficacité d'élagage de LC. Dans le meilleur des cas, un solveur qui maintient un niveau de cohérence intermédiaire p-LC explore le même nombre de nœuds qu'en maintenant LC avec un nombre de vérification de contraintes aussi réduit que lors du maintien d'AC, ce qui entraine une baisse de temps CPU de résolution comparé aux temps obtenus utilisant AC ou LC.

2 Préliminaires

Un *réseau de contraintes* est défini par un ensemble de n variables $X = \{x_1, ..., x_n\}$, un ensemble de domaines ordonnés $D = \{D(x_1), ..., D(x_n)\}$ où $D(x_i)$ est l'ensemble de valeurs possibles pour la variable x_i, et un ensemble de e contraintes $C = \{c_1, ..., c_e\}$. Chaque contrainte c_k est définie par une paire $(var(c_k), sol(c_k))$, où $var(c_k)$ est un sous-ensemble ordonné de X, et $sol(c_k)$ est l'ensemble de combinaisons des valeurs (uplets) que peuvent prendre simultanément les variables de $var(c_k)$. Une contrainte peut être spécifiée en extension par la liste des uplets la satisfaisant, ou en intention par une formule mathématique qui est la fonction caractéristique de la contrainte. Dans la suite, nous nous limitons aux contraintes binaires car certaines propriétés de cohérence locale que nous utilisons sont définies uniquement dans le cas binaire. Cependant, les notions que nous introduisons peuvent être étendues au cas non-binaires. Nous utilisons la notation c_{ij} pour désigner la contrainte impliquant x_i et x_j, et $\Gamma(x_i)$ pour désigner l'ensemble des variables x_j impliquées avec x_i dans une contrainte.

Une valeur $v_j \in D(x_j)$ est appelée *support arc-consistant (support AC)* de $v_i \in D(x_i)$ sur c_{ij} si le tulpe $(v_i, v_j) \in sol(c_{ij})$. Une valeur $v_i \in D(x_i)$ est dite *arc consistante (AC)* si et seulement si pour toute variable $x_j \in \Gamma(x_i)$, v_i possède un support AC $v_j \in D(x_j)$ sur c_{ij}. Un domaine $D(x_i)$ est AC s'il n'est pas vide et toutes les valeurs dans $D(x_i)$ sont AC. Un réseau est AC si tous les domaines dans D sont AC. Si la réalisation de la cohérence d'arc dans un réseau de contraintes N conduit à un domaine vide, on dit que N est arc inconsistant.

Un uplet $(v_i, v_j) \in D(x_i) \times D(x_j)$ est *consistant de chemin (PC)* si et seulement si pour toute variable tierce x_k il existe une valeur $v_k \in D(x_k)$ telle que v_k est un support AC de v_i et v_j. Dans ce cas, v_k est appelée *témoin* de la cohérence de chemin pour (v_i, v_j).

Une valeur $v_j \in D(x_j)$ est appelée *support maxRPC* de $v_i \in D(x_i)$ sur c_{ij} si et seulement si v_j est un support AC de v_i sur c_{ij} et le uplet (v_i, v_j) est consistant de chemin. Une valeur $v_i \in D(x_i)$ est maxRPC sur une contrainte c_{ij} si et seulement si $\exists v_j \in D(x_j)$ support maxRPC de v_i sur c_{ij}. Une valeur $v_i \in D(x_i)$ est maxRPC si et seulement si, pour toute variable $x_j \in \Gamma(x_i)$ v_i possède un support maxRPC $v_j \in D(x_j)$ sur c_{ij}. Un domaine $D(x_i)$ est maxRPC s'il n'est pas vide et toutes les valeurs dans $D(x_i)$ sont maxRPC. Un réseau est maxRPC si tous les domaines dans D sont maxRPC.

Étant donné un réseau de contraintes $N = (X, D, C)$, une valeur $v_i \in D(x_i)$ est dite *Singleton Arc consistante (SAC)* si et seulement si le réseau $N|_{x_i=v_i}$ est arc consistant. Un domaine $D(x_i)$ est SAC s'il n'est pas vide et toutes les valeurs dans $D(x_i)$ sont SAC. Un réseau est SAC si tous les domaines dans D sont SAC.

Une propriété de cohérence locale LC_1 est dite plus forte (ou plus puissante) qu'une autre propriété de cohérence locale LC_2 ($LC_2 \preceq LC_1$) si LC_2 est vérifiée dans tout réseau de contraintes où LC_1 est vérifiée, et qu'il existe des réseaux de contraintes où LC_2 est vérifiée mais pas LC_1.

3 La Cohérence Paramétrée

Dans cette section nous présentons une approche originale qui permet de paramétrer une propriété de

cohérence locale LC plus forte que la cohérence d'arc. La force (puissance) de cette cohérence paramétrée est fonction d'un paramètre p, elle dégénère à la cohérence d'arc lorsque le paramètre est égal à 0, à la cohérence forte LC lorsque le paramètre est égal à 1, et se positionne entre les deux lorsque le paramètre est compris entre 0 et 1. L'idée derrière cela est d'être capable d'ajuster le niveau de cohérence locale en fonction de l'instance du problème à résoudre en espérant qu'un tel niveau de cohérence adapté supprime nettement plus de valeurs inconsistantes que la cohérence d'arc tout en ayant un coût en terme de temps d'exécution inférieur au coût de maintien de LC.

La cohérence paramétrée est basée sur le concept de la stabilité des valeurs. Nous avons besoin de définir un concept qui évalue combien une valeur est loin d'être la dernière dans son domaine. Dans ce qui suit, $rank(v, S)$ désigne la position de la valeur v dans l'ensemble ordonné de valeurs S.

Définition 1 (Distance à la fin pour une valeur) *Pour une valeur $v_i \in D(x_i)$, la distance à la fin est donnée par :*

$$\Delta(x_i, v_i) = (|D_o(x_i)| - rank(v_i, D_o(x_i)))/|D_o(x_i)|,$$

où $D_o(x_i)$ est le domaine initial de x_i

Nous pouvons observer que la première valeur de $D_o(x_i)$ a la distance $(|D_o(x_i)| - 1)/|D_o(x_i)|$, et la dernière valeur a la distance 0. Ainsi, $\forall v_i \in D(x_i), 0 \leq \Delta(x_i, v_i) < 1$.

Nous pouvons maintenant donner la définition de ce que nous appelons la stabilité paramétrée d'une valeur pour la cohérence d'arc.

Définition 2 (p-stabilité pour AC) *Une valeur $v_i \in D(x_i)$ est p-stable pour AC sur c_{ij} si et seulement si v_i possède un support AC $v_j \in D(x_j)$ sur c_{ij}, tel que $\Delta(x_j, v_j) \geq p$. une valeur $v_i \in D(x_i)$ est p-stable pour AC si et seulement si $\forall x_j \in \Gamma(x_i)$, v_i est p-stable pour AC sur c_{ij}.*

Nous sommes maintenant prêts pour donner une première définition de la cohérence locale paramétrée. Cette première définition peut être appliquée sur n'importe quelle cohérence locale LC pour qui la cohérence d'une valeur sur une contrainte est bien définie. C'est le cas par exemple pour toutes les cohérences dites triangle-based [2].

Définition 3 (Constraint-based p-LC) *Soit LC une cohérence locale plus forte que l'AC pour qui la cohérence d'une valeur sur une contrainte est définie. Une valeur $v_i \in D(x_i)$ est constraint-based p-LC sur c_{ij} si et seulement si elle est p-stable pour AC sur c_{ij}, ou elle est LC sur c_{ij}. Une valeur $v_i \in D(x_i)$ est constraint-based p-LC si et seulement si $\forall c_{ij}$, v_i est constraint-based p-LC sur c_{ij}. Un réseau de contraintes est constraint-based p-LC, si et seulement si toutes les valeurs dans tous les domaines dans D sont constraint-based p-LC.*

Théorème 4 *Soit LC une cohérence locale plus forte que l'AC pour qui la cohérence d'une valeur sur une contrainte est définie. Soit p_1 et p_2 deux paramètres dans [0..1]. Si $p_1 < p_2$ alors AC $\preceq$ constraint-based p_1-LC $\preceq$ constraint-based p_2-LC $\preceq$ LC.*

Preuve. Supposons qu'il existe deux paramètres p_1 et p_2 tels que $0 \leq p_1 < p_2 \leq 1$, et supposons qu'il existe un réseau de contrainte N qui est p_2-LC et qui contient une valeur (x_i, v_i) qui est p_1-LC inconsistante. Soit c_{ij} la contrainte sur laquelle (x_i, v_i) est p_1-LC inconsistante. Donc,

1. $\nexists v_j \in D(x_j)$ support AC de (x_i, v_i) sur c_{ij} tel que $\Delta(x_j, v_j) \geq p_1$. Alors, v_i n'est pas p_2-stable pour AC sur c_{ij}.
2. v_i n'est pas LC sur c_{ij}.

(1) et (2) impliquent que v_i n'est pas p_2-LC, et N n'est pas p_2-LC. ∎

Pour les cohérences pour lesquelles la cohérence d'une valeur sur une contrainte n'est pas définie, nous donnons une deuxième définition de la cohérence paramétrée.

Définition 5 (Value-based p-LC) *Soit LC une cohérence locale plus forte que l'AC. Une valeur $v_i \in D(x_i)$ est value-based p-LC si et seulement si elle est p-stable pour AC ou elle est LC. Un réseau de contraintes est value-based p-LC si et seulement si toutes les valeurs dans tous les domaines dans D sont value-based p-LC.*

Théorème 6 *Soit LC une cohérence locale plus forte que AC. Soit p_1 et p_2 deux paramètres dans [0..1]. Si $p_1 < p_2$ alors AC $\preceq$ value-based p_1-LC $\preceq$ value-based p_2-LC $\preceq$ LC.*

Preuve. Supposons qu'il existe deux paramètres p_1, p_2 tel que $0 \leq p_1 < p_2 \leq 1$, et supposons qu'il existe un réseau de contraintes N qui est p_2-LC et qui contient une valeur (x_i, v_i) qui est p_1-LC inconsistante. v_i est p_1-LC incohérence signifie que :

1. v_i n'est pas p_1-stable pour AC : $\exists c_{ij}$ où v_i n'est pas p_1-stable pour AC. Donc $\nexists v_j \in D(x_j)$ support AC de (x_i, v_i) sur c_{ij} tel que $\Delta(x_j, v_j) \geq p_1$. Alors, v_i n'est pas p_2-stable pour AC sur c_{ij}, donc v_i n'est pas p_2-stable pour AC.
2. v_i est LC inconsistante.

(1) et (2) impliquent que v_i n'est pas p_2-LC, et donc N n'est pas p_2-LC. ∎

Pour les deux définitions de p-LC, nous avons la propriété suivante pour les cas extrêmes (p=0, p=1).

Corollaire 7 *Soit LC_1 et LC_2 deux propriétés de cohérence locale plus fortes que l'AC, telles que la cohérence LC_1 d'une valeur sur une contrainte est définie. Nous avons : constraint-based 0-LC_1 = AC et constraint-based 1-LC_1 = LC_1. value-based 0-LC_2 = AC et value-based 1-LC_2 = LC_2.*

4 maxRPC Paramétrée : p-maxRPC

Dans cette section nous appliquons le concept de cohérence paramétrée à maxRPC pour obtenir p-maxRPC, une cohérence qui permet d'atteindre des niveaux de cohérence intermédiaires entre AC et maxRPC.

Définition 8 (p-maxRPC) *Une valeur, un réseau sont p-maxRPC si et seulement si ils sont constraint-based p-maxRPC.*

A partir du théorème 4 et du corollaire 7 nous obtenons le corollaire suivant.

Corollaire 9 *Pour tous paramètres p_1 et p_2 tels que $0 \leq p_1 < p_2 \leq 1$, $AC \preceq p_1$-$maxRPC \preceq p_2$-$maxRPC \preceq maxRPC$. 0-$maxRPC = AC$ et 1-$maxRPC = maxRPC$.*

Dans ce qui suit, nous décrivons uniquement les changements que nous avons apportés à l'algorithme maxRPC3 pour concevoir p-maxRPC3, un algorithme qui réalise p-maxRPC.

maxRPC3 utilise une liste de propagation Q qui contient les variables dont le domaine a changé. Il utilise aussi deux autres structures de données : LastAC et LastPC. Pour chaque valeur (x_i, v_i) $\text{LastAC}_{x_i,v_i,x_j}$ est utilisé pour stocker le plus petit support AC de (x_i, v_i) sur c_{ij} et $\text{LastPC}_{x_i,v_i,x_j}$ est utilisé pour stocker le plus petit support PC de (x_i, v_i) sur c_{ij} (c-à-d. le plus petit support AC (x_j, v_j) de (x_i, v_i) sur c_{ij} tel que (v_i, v_j) est PC). Cet algorithme consiste en deux phases : une phase d'initialisation et une phase de propagation.

Dans la phase d'initialisation maxRPC3 vérifie si chaque valeur (x_i, v_i) possède un support maxRPC (x_j, v_j) sur chaque contrainte c_{ij}. Sinon, il supprime v_i de $D(x_i)$ et insère x_i dans Q. Pour vérifier si une valeur (x_i, v_i) possède un support maxRPC sur une contrainte c_{ij}, maxRPC3 recherche d'abord un support AC (x_j, v_j) de (x_i, v_i) sur c_{ij}, puis vérifie si (v_i, v_j) est PC. Dans cette dernière étape nous avons modifié maxRPC3 et nous ne vérifions la cohérence de chemin de (v_i, v_j) que si la distance $\Delta(x_j, v_j)$ est inférieure au seuil p.

La phase de propagation consiste à propager l'effet des suppressions. Tant que Q contient des variables, maxRPC3 extrait une variable x_i de Q et vérifie pour chaque valeur (x_j, v_j) de chaque variable voisine $x_j \in \Gamma(x_i)$ si elle n'est pas devenue maxRPC-inconsistante à cause des suppressions de valeurs dans $D(x_i)$.

Dans cette phase aussi, nous avons modifié maxRPC3 pour vérifier si les valeurs sont encore p-maxRPC au lieu de vérifier si elles sont maxRPC. Dans p-maxRPC3, le plus petit support p-maxRPC de (x_j, v_j) sur c_{ij} correspond au plus petit support AC si (x_j, v_j) est p-stable pour AC sur c_{ij}. Sinon, il correspond au plus petit support PC. Ainsi, p-maxRPC3 vérifie si le dernier support p-maxRPC (le dernier support connu) de (x_j, v_j) sur c_{ij} appartient toujours au domaine de x_i. Sinon, il cherche le prochain support AC (x_i, v_i) sur c_{ij}, et vérifie si (v_i, v_j) est PC dans le cas où $\Delta(x_i, v_i) < p$. Si aucun support p-maxRPC n'existe, p-maxRPC3 supprime la valeur et insère la variable x_j dans la liste Q. Si la valeur (x_j, v_j) n'a pas était supprimée dans la phase précédente, p-maxRPC3 vérifie la subsistance du témoin (x_i, v_i) pour chaque paire (v_j, v_k) tel que (x_k, v_k) est le support p-maxRPC de (v_j) sur c_{jk} et $\Delta(x_k, v_k) < p$. Sinon, il recherche le prochain support p-maxRPC de (v_j) sur c_{jk}, et supprime (x_j, v_j) de $D(x_j)$ si ce support n'existe pas puis insère la variable x_j dans Q.

5 SAC Paramétrée : p-SAC

Cette section présente la singleton arc cohérence paramétrée (p-SAC), une instanciation du concept générique de cohérence paramétrée à la la singleton arc cohérence, pour qui la cohérence d'une valeur sur une contrainte n'est pas définie.

Définition 10 (p-SAC) *Une valeur, un réseau, sont p-SAC si et seulement s'ils sont value-based p-SAC.*

A partir du théorème 6 et du corollaire 7 nous obtenons le corollaire suivant.

Corollaire 11 *Pour tous paramètres p_1 et p_2 tels que $0 \leq p_1 < p_2 \leq 1, AC \preceq p_1$-$SAC \preceq p_2$-$SAC \preceq SAC$. 0-$SAC = AC$ et 1-$SAC = SAC$.*

Nous proposons maintenant un algorithme pour réaliser la singleton arc cohérence paramétrée. Sa fonction principale est décrite dans Algorithme 1. Elle commence par un appel à la fonction `InstrumentedAC`, qui applique simplement la cohérence d'arc et insère dans la liste P toutes les valeurs (x_i, v_i) qui ne sont

Algorithm 1: p-SAC1(X, D, C)

```
1  begin
2      P ← ∅
3      if ¬InstrumentedAC(X, D, C, P) then
4          return false
5      while P ≠ ∅ do
6          Change ← false
7          if ¬CustomizedSAC(X, D, C, P, Change) then
8              return false;
9          if Change then
10             foreach x_i ∈ X, v_i ∈ D(x_i) do
11                 if v_i is not p-stable for AC then
12                     P ← P ∪ (x_i, v_i)
13     return true
```

Algorithm 2: CustomizedSAC($X, D, C, P, Change$)

```
1  begin
2      while P ≠ ∅ do
3          pick and delete (x_i, v_i) from P
4          if v_i ∈ D(x_i) ∧ ¬AC(X, D, C ∪ {x_i = v_i}) then
5              remove v_i from D(x_i)
6              Change ← true
7              if D(x_i) = ∅ or ¬propagateAC(X, D, C, {x_i})
               then
8                  return false
9      return true
```

pas p-stable pour AC (ligne 3). Ensuite, elle entre dans une boucle. Dans la première ligne de la boucle (ligne 7), elle fait appel à la procédure `CustomizedSAC` qui vérifie SAC pour chaque valeur (x_i, v_i) dans P, et supprime v_i de $D(x_i)$ si elle est SAC-inconsistante. Chaque fois qu'une valeur SAC-inconsistante est supprimée, le flag $Change$ est positionné à vrai et la cohérence d'arc est rétablie à nouveau dans le réseau de contraintes. Quand `CustomizedSAC` n'échoue pas et retourne $Change = vrai$, toutes les valeurs qui ne sont plus p-stables pour AC sont mises dans P pour vérifier si elles sont SAC dans l'itération suivante. L'algorithme termine quand un domaine vide est détecté dans `InstrumentedAC` ou `CustomizedSAC`, ou quand `CustomizedSAC` retourne $Change = faux$, ou quand `InstrumentedAC` retourne $vrai$ et P vide.

La procédure `CustomizedSAC`, présentée dans Algorithme 2, procède comme suit : tant que P n'est pas vide, elle prend une valeur (x_i, v_i) de P (ligne 3). Si la valeur v_i est encore dans $D(x_i)$, `CustomizedSAC` vérifie si v_i est SAC (ligne 4) et la supprime de $D(x_i)$ si elle est SAC-inconsistante (ligne 5). Quand v_i est supprimée, le flag Change est positionné à vrai (ligne 6) puis la procédure `propagateAC` est appelée pour rétablir la cohérence d'arc (ligne 7). Si un domaine vide est détecté, la procédure `CustomizedSAC` retourne faux (ligne 8). Elle retourne vrai quand P devient vide (ligne 9).

6 Évaluation Expérimentale

Nous avons implémenté les algorithmes réalisant p-maxRPC et p-SAC comme décrits dans la section précédente, en plus de maxRPC3, SAC1 et AC2001 [3]. Tous ces algorithmes sont implémentés dans notre propre solveur de contraintes binaires et sont maintenus pendant la recherche. Nous avons testé ces algorithmes dans plusieurs classes de problèmes de la dernière compétition internationale de solveurs de contraintes 09 [1]. Nous avons sélectionné les problèmes contenant uniquement des contraintes binaires. Pour résoudre un problème utilisant un niveau de cohérence p-LC, cette cohérence est d'abord établie dans une étape de prétraitement, puis maintenue pendent la recherche. Pour isoler l'effet de la propagation, nous utilisons l'ordre lexicographique pour les variables et les valeurs. Nous avons fixé une heure comme limite de temps CPU. Nous avons réalisé nos expérimentations sur une plateforme 12-core sur une machine GenuineIntel avec 16 Go de RAM fonctionnant à 2.92 GHz.

Nous avons expérimenté le concept de cohérence paramétrée p-LC dans les cas où LC correspond à maxRPC ou SAC, et nous l'avons utilisé pour résoudre des instances de problèmes en maintenant les différents niveaux intermédiaires de cohérence p-LC, allant de la cohérence d'arc (0-LC) à la cohérence forte (1-maxRPC ou 1-SAC), en variant p de 0 à 1 avec un pas de 0.1. Pour chaque instance résolue, nous avons enregistré le paramètre associé au meilleur temps CPU obtenu. Les performances ont été mesurées en termes de temps CPU en secondes, de nombre de nœuds visités (NODE) et de nombre de contraintes testées (CCK). Pour les deux cohérences expérimentées (maxRPC et SAC), les résultats sont données sous la forme "temps CPU(p)", où (p) est le paramètre pour lequel p-LC a donnée le meilleur résultat.

Les tables 1, 2 et 3 comparent les performances de maintien de p-maxRPC et p-SAC aux performances de AC, maxRPC et SAC sur les instances du problème d'allocation de fréquences de liaison radio (RLFAP), les instances du problème Geom, et les instances du problème des Queens-Knights. La comparaison de p-maxRPC à maxRPC et AC dans les problèmes RLFAP et Geom montre, pour la plupart des instances de ces problèmes, l'existence d'un paramètre p où la résolution avec p-maxRPC est plus rapide que AC et

1. http ://cpai.ucc.ie/09/

TABLE 1 – Performances (temps cpu, nœuds et ccks) de p-maxRPC et p-SAC sur les instances du problème RLFAP.

		AC	p-maxRPC	maxRPC	p-SAC	SAC
scen1-f8	cpu(s)	time-out	**1.39(0.2)**	6.1	time-out	time-out
	#nodes	-	927	917	-	-
	#ccks	-	1397440	26932990	-	-
scen2-f24	cpu(s)	time-out	**0.13(0.3)**	0.65	52.77(0.1)	531.29
	#nodes	-	201	201	202	200
	#ccks	-	296974	3462070	8751216	170233715
scen3-f10	cpu(s)	time-out	**0.89(0.5)**	2.8	2162.94(0.5)	time-out
	#nodes	-	469	408	407	-
	#ccks	-	874930	13311797	161605804	-
scen6-w1	cpu(s)	**0.02**	**0.02(0.2)**	0.08	0.09(0.0)	259.74
	#nodes	211	211	211	211	200
	#ccks	295128	296541	904900	295128	233056049
scen6-w2	cpu(s)	0.11	**0.01(1.0)**	**0.01**	0.1(0.7)	0.18
	#nodes	40	0	0	0	0
	#ccks	411071	85769	85769	417870	431405
scen7-w1-f4	cpu(s)	0.08	**0.06(0.2)**	0.14	0.31(0.0)	983.9
	#nodes	424	419	406	424	400
	#ccks	509989	559375	1319246	509989	209684190
scen7-w1-f5	cpu(s)	time-out	**0.04(0.2)**	0.08	1.3(0.2)	9.36
	#nodes	-	0	0	0	0
	#ccks	-	478795	1087223	806395	2020584
scen10-w1-f3	cpu(s)	time-out	**0.05(0.5)**	0.1	3.37(0.2)	13.27
	#nodes	-	0	0	0	0
	#ccks	-	1040434	1714477	1377050	2764334

TABLE 2 – Performances (temps cpu, nœuds et ccks) de p-maxRPC et p-SAC sur les instances du problème Geom

		AC	p-maxRPC	maxRPC	p-SAC	SAC
geo50-20-d4-75-26	cpu(s)	111.48	17.8(1.0)	15.07	**3.19(0.7)**	13.33
	#nodes	477696	3768	3768	51	51
	#ccks	96192822	40784017	40784017	3336287	9304263
geo50-20-d4-75-30	cpu(s)	1.4	**0.14(0.9)**	0.24	0.82(0.1)	10.95
	#nodes	5098	55	55	59	50
	#ccks	1163409	351258	894200	196147	9709063
geo50-20-d4-75-38	cpu(s)	1800.13	**779.01(0.6)**	1017.08	1464.56(0.3)	1901.09
	#nodes	3870476	314454	166210	4118134	4582
	#ccks	1528095910	1445180475	2494969079	1080827574	1350511787
geo50-20-d4-75-43	cpu(s)	1671.35	**1264.36(0.5)**	1530.02	2546.05(0.0)	time-out
	#nodes	4118134	555259	279130	4118134	-
	#ccks	1160664461	1801402535	3898964831	1160664461	-
geo50-20-d4-75-46	cpu(s)	1732.22	**371.3(0.6)**	517.35	1576.13(0.1)	time-out
	#nodes	3682394	125151	64138	253045	-
	#ccks	1516856615	584743023	1287674430	851048532	-
geo50-20-d4-75-78	cpu(s)	**0.13**	0.22(0.6)	0.31	0.17(0.0)	9.77
	#nodes	389	122	84	389	55
	#ccks	170210	351104	1077835	170210	7951262
geo50-20-d4-75-84	cpu(s)	404.63	0.44(0.6)	0.56	**0.41(0.1)**	10.18
	#nodes	2581794	513	333	206	50
	#ccks	293092144	800657	1606047	444784	7887350

maxRPC. Nous constatons la même chose en comparant les performances de p-SAC à SAC et AC sur les instances des problèmes RLFAP et Geom. En général, pour la plupart des instances des problèmes expérimentés, à l'exception du problème Queens-Knights, nous constatons l'existence d'un paramètre p où p-LC est plus rapide que AC et LC. En fait, chaque fois que p-LC améliore AC et LC, il parvient à trouver un compromis entre le nombre de nœuds visités (la puissance de la cohérence p-LC) et le nombre de CCK produit en maintenant p-LC.

Les Figures 1 et 2 illustrent les performances (CPU, NODE et CCK) de p-maxRPC pour des valeurs de p allant de 0 à 1 avec un pas de 0.1. La Figure 1 montre un exemple où le maintien de p-maxRPC résout le problème plus rapidement que AC et maxRPC pour les valeurs de p allant de 0.3 à 0.8. Nous observons que p-maxRPC est plus rapide que AC et maxRPC lorsqu'il réduit la taille de l'espace de recherche aussi bien que maxRPC (même nombre de nœuds visités) avec un nombre de CCK proche du nombre de CCK produit par AC. La Figure 2 montre une instance où p-maxRPC échoue à améliorer le temps CPU de AC et maxRPC en même temps. Elle nous montre que, sauf pour p=1, p-maxRPC est deux à trois fois plus rapide que maxRPC. p-maxRPC améliore maxRPC car il arrive à réduire considérablement le nombre de CCK. Cependant, il n'a pas pu améliorer AC car le nombre de CCK produit par p-maxRPC reste élevé comparé au nombre de CCK produit par AC, alors que la réduction du nombre de nœuds visités n'est pas significative comparée au nombre de nœuds visités par AC.

Pour les deux figures 1 et 2 nous observons que le temps CPU pour 1-maxRPC (respectivement 0-maxRPC) est supérieur au temps CPU pour maxRPC (respectivement AC), bien que les deux cohérences sont équivalentes. En général p-LC est légèrement moins performante que LC (respectivement AC) pour p=1 (respectivement p=0) car elle effectue des tests et des calculs supplémentaires liés à la distance des valeurs. Pour p=0, cette différence est également explicable par le fait que p-LC maintient des structures de données que AC n'utilise pas.

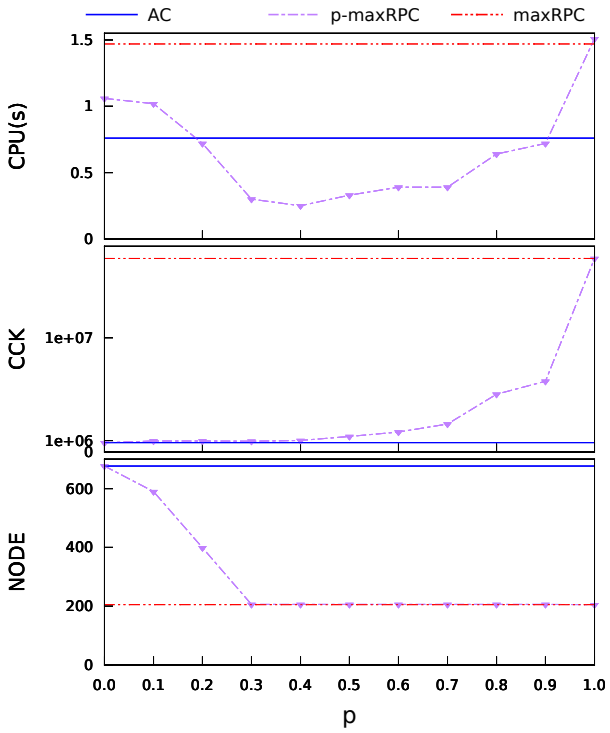

FIGURE 1 – Instance où p-maxRPC améliore AC et maxRPC

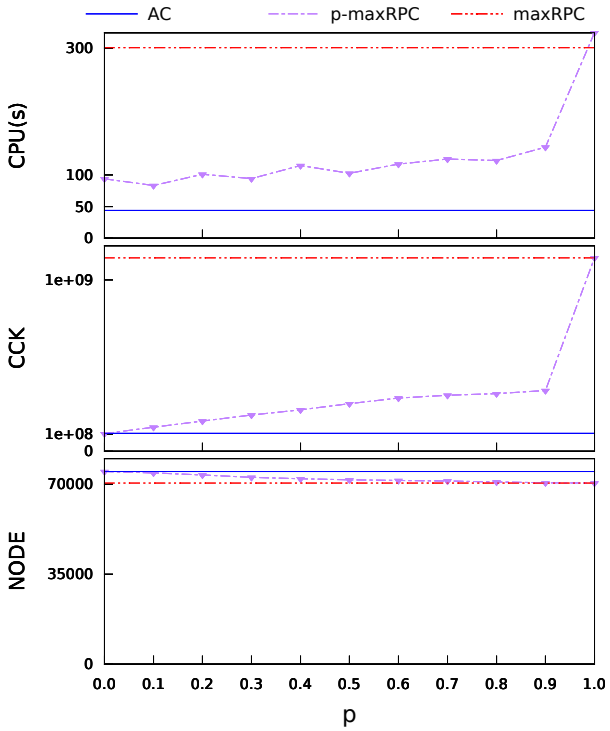

FIGURE 2 – Instance où p-maxRPC échoue à améliorer AC

7 Conclusion

Nous avons introduit la notion de *stabilité* des valeurs pour la cohérence d'arc, une notion basée sur la profondeur des supports AC de ces valeurs dans leur domaine. Nous avons utilisé cette notion pour proposer la *cohérence paramétrée*, une technique qui permet de définir des niveaux de cohérence locale intermédiaires entre la cohérence d'arc et une cohérence locale plus

TABLE 3 – Performances (temps cpu, nœuds et ccks) de p-maxRPC et p-SAC sur les instances du problème Queens-Knights

		AC	p-maxRPC	maxRPC	p-SAC	SAC
queensKnights-10-5-add	cpu(s)	27.14	30.79(0.2)	98.44	0.09(1.0)	**0.06**
	#nodes	82208	81033	78498	0	0
	#ccks	131098933	148919686	954982880	235110	235110
queensKnights-10-5-mul	cpu(s)	43.89	83.27(0.1)	300.74	0.34(1.0)	**0.2**
	#nodes	74968	74414	70474	0	0
	#ccks	104376698	140309576	1128564278	432537	432537
queensKnights-15-5-add	cpu(s)	time-out	time-out	time-out	0.56(1.0)	**0.32**
	#nodes	-	-	-	0	0
	#ccks	-	-	-	1172468	1172468
queensKnights-8-5-add	cpu(s)	1.08	1.48(0.2)	4.48	**0.03(1.0)**	0.04
	#nodes	6656	6629	6502	0	0
	#ccks	6205191	7104271	44551267	94216	94216
queensKnights-8-5-mul	cpu(s)	1.77	3.08(0.1)	11.15	0.11(1.0)	**0.05**
	#nodes	5920	5920	5638	0	0
	#ccks	4599166	5194507	46125818	174627	174627

forte que l'AC. La force (puissance) des niveaux intermédiaires résultants est croissante et est fonction du paramètre p. Nous avons instancié l'approche générique de cohérence paramétrée en utilisant la max-cohérence de chemin restreinte et la singleton arc cohérence. Nous avons montré expérimentalement sur de nombreux exemples de problèmes que le concept de cohérence paramétrée est viable. Notre prochain objectif est d'être en mesure d'adapter le paramètre au cours de la recherche pour n'importe quelle instance de problème. Grâce aux techniques d'apprentissage automatique, le solveur doit être capable de sélectionner la cohérence d'arc ou une cohérence locale plus forte, en fonction de l'instance du problème à résoudre, en fonction de la partie du problème dans laquelle s'effectue la propagation et en fonction de l'évolution de l'arbre de recherche développé par le solveur. Cette méthode générique devrait être valable pour toute propriété de cohérence locale et pour n'importe quel type de problème.

Références

[1] Thanasis Balafoutis, Anastasia Paparrizou, Kostas Stergiou, and Toby Walsh. New algorithms for max restricted path consistency. *Constraints*, 16(4) :372–406, 2011.

[2] Christian Bessiere. Constraint propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 3. Elsevier, 2006.

[3] Christian Bessière, Jean-Charles Régin, Roland H. C. Yap, and Yuanlin Zhang. An optimal coarse-grained arc consistency algorithm. *Artif. Intell.*, 165(2) :165–185, 2005.

[4] Romuald Debruyne and Christian Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *IJCAI (1)*, pages 412–417. Morgan Kaufmann, 1997.

[5] Alan K. Mackworth. Consistency in networks of relations. *Artif. Intell.*, 8(1) :99–118, 1977.

[6] Kostas Stergiou. Heuristics for dynamically adapting propagation in constraint satisfaction problems. *AI Commun.*, 22 :125–141, August 2009.