Calcul d'approximations intérieures pour la résolution de Max-NCSP

Marc Christie, Jean-Marie Normand, Charlotte Truchet

Laboratoire d'Informatique de Nantes Atlantique
FRE CNRS 2729, Université de Nantes
2 Rue de la Houssinière, 44322 Nantes Cedex 3
{christie,normand,truchet}@lina.univ-nantes.fr

Abstract

Cet article présente un algorithme de calcul de solutions garanties pour des problèmes numériques MaxCSP. Les auteurs proposent une approche qui offre une exploration complète et garantie de l'espace de recherche et identifie l'ensemble des sous-espaces éventuellement disjoints qui satisfont le plus grand nombre de contraintes numériques du CSP. L'implémentation de cet algorithme fournit une approximation intérieure de ces sous-ensembles pour une taille minimale de pavé donnée. L'approche repose sur des méthodes de propagation classiques sur les intervalles (AC-3) et sur un processus d'extension par l'intérieur en n dimensions. Une hybridation avec des méthodes locales de recherche, étendues aux intervalles, est proposée et testée. Les résultats, portant sur des problèmes *jouets* montrent l'apport par rapport à un algorithme naïf mais ne mettent pas en lumière l'évidence de la contribution des méthodes de recherche locale.

1 Introduction

Les approches complètes de résolution de CSP numériques permettent de s'attaquer à des problèmes réputés difficiles dans de nombreux domaines (conception préliminaire, conformation moléculaire, problèmes d'automatique, de robotique, ou de synthèse d'images, *etc.*). Ces méthodes déterminent un ensemble de pavés approximant par l'extérieur l'ensemble réel de solutions assurant qu'aucune solution réelle n'est perdue lors du processus de résolution (propriété de complétude). Or de nombreuses applications nécessitent le calcul de solutions sûres, c'est-à-dire telles que tout point des pavés résultats soit une solution garantie du système de contraintes. De tels problèmes ont été résolus par différentes approches [24, 15, 5] dites de calcul d'approximation intérieure ou *i-consistency* [10]. Cependant, aucune de ces deux approximations (intérieure ou extérieure) n'est directement adaptée au cadre des problèmes Max-NCSP qui cherche à maximiser le nombre de contraintes satisfaites dans un CSP sur les domaines continus (ici, on n'établit pas de hiérarchie entre les contraintes ou de préférences sur les domaines). Ce travail est motivé par le fait qu'il est généralement difficile de déterminer à l'avance la nature d'un modèle numérique (*i.e.* sous-contraint ou sur-contraint) avant sa résolution, et donc de prévoir la nature des techniques à employer pour le résoudre.

Nous proposons dans cet article les premiers résultats issus d'un algorithme de calcul d'approximations intérieures pour des problèmes sur ou sous-contraints. Cet algorithme détermine un ensemble de pavés qui satisfont un maximum de contraintes du CSP. La technique de résolution consiste à entrelacer des étapes de propagation de contraintes avec des étapes d'extensions intérieures basées sur le choix d'un point représentatif d'un pavé. Cette sélection sera effectuée par deux techniques différentes, la première étant un simple tirage du milieu du pavé (midPoint), la deuxième faisant appel à une méthode de Recherche Locale (LS) étendue au domaine continu pour aider à déterminer le meilleur tirage possible dans le pavé. À ce sujet, nous définissons un cadre formel pour l'extension au continu de la recherche locale. Nous proposons de comparer les résultats de l'algorithme dit de midPoint avec un simple processus de bissection/évaluation, puis de les comparer aux solutions obtenues par l'approche de recherche locale continue. A la

connaissance des auteurs, il n'existe pas dans la littérature d'approches Max-NCSP pour le calcul d'approximations intérieures sur des systèmes numériques.

L'article s'organise de la façon suivante. La section 2 présente les fondements liés aux techniques de résolution de CSP numériques puis propose, à l'instar des opérateurs de contraction $n-$aires extérieurs, un opérateur simple d'extension $n-$aire intérieur. La section 3 définit ensuite la classe des problèmes Max-NCSP, et propose une méthode de résolution basée sur l'application d'opérateurs de contraction et d'extension. Nous définissons ensuite un cadre pour une extension au continu de la recherche locale dans la section 4, avant de présenter quelques résultats.

2 Résolution par Contraintes d'Intervalles

2.1 L'analyse par intervalles

Dans cette section, nous limiterons la présentation de l'analyse d'intervalles aux concepts utilisés dans la suite du papier. Pour une présentation plus approfondie de l'analyse d'intervalles, le lecteur peut se référer à [21, 1, 12, 22]. Une solution proposée par Moore [21] pour contrôler les erreurs d'arrondis rencontrées lors des calculs en arithmétique flottante est l'utilisation de l'*arithmétique des intervalles*. Celle-ci consiste à remplacer les nombres réels ($\in \mathbb{R}$) par des intervalles les contenant et dont les bornes sont représentables en machine. Un intervalle I dont les bornes sont représentables est appelé un *intervalle de flottants* et est de la forme : $I = [a, b] = \{r \in \mathbb{R} \mid a \leq r \leq b$, avec $a, b \in \mathbb{F}\}$ où $\mathbb{F}$ représente l'ensemble de tous les flottants. Un intervalle non-vide $I = [a, b]$ qui contient au plus deux nombres flottants est dit *canonique*. De la même manière, un produit Cartésien d'intervalles (aussi appelé *boîte* ou *pavé*) est dit canonique lorsqu'il est canonique dans chacune de ses dimensions. Soit $\mathcal{I}$ l'ensemble contenant tous les intervalles de nombres flottants. Dans la suite de ce papier, les produits Cartésiens et les vecteurs seront notés en gras.

Les opérations et les fonctions sur les réels sont dès lors remplacées par une *extension aux intervalles* possédant la propriété dite de *containment* (*cf.* [21]). Soit une fonction sur les réels, $f : \mathbb{R}^n \to \mathbb{R}$ et une boîte $\boldsymbol{B} \in \mathcal{I}^n$, soit $f(\boldsymbol{B}) = \{f(\boldsymbol{r}) \mid \boldsymbol{r} \in \boldsymbol{B}\}$, $\mathcal{D}_f$ le domaine de f, et $\mathcal{D}_f^{\mathcal{I}} = \{\boldsymbol{B} \in \mathcal{I}^n \mid \boldsymbol{B} \subseteq \mathcal{D}_f\}$.

Définition 1 (Extension aux intervalles [22]) *Une* extension aux intervalles $F : \mathcal{I}^n \to \mathcal{I}$ *de* f *est une fonction sur les intervalles vérifiant les propriétés suivantes :*

$$f(\boldsymbol{r}) = F(\boldsymbol{r}) \qquad \forall \boldsymbol{r} \in \mathcal{D}_f$$
$$\mathsf{Outer}(f(\boldsymbol{B})) \subseteq F(\boldsymbol{B}) \qquad \forall \boldsymbol{B} \in \mathcal{D}_f^{\mathcal{I}}$$

avec $\mathsf{Outer}(\rho) = \bigcap\{\boldsymbol{B} \in \mathcal{I}^n \mid \rho \subseteq \boldsymbol{B}\}$ *pour toute relation réelle* $\rho \in \mathbb{R}^n$.

Les extensions aux intervalles de quelques-uns des opérateurs basiques sur les réels sont définies comme suit :

$$\begin{aligned} [a,b] \oplus [c,d] &= [a+c, b+d] \\ [a,b] \ominus [c,d] &= [a-d, b-c] \\ [a,b] \otimes [c,d] &= [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)] \\ \exp([a,b]) &= [\exp(a), \exp(b)] \end{aligned}$$

2.2 Approximations extérieures

Éliminer toutes les valeurs inconsistantes d'une boîte est un problème insoluble pour un système de contraintes réelles. Par conséquent, des méthodes telles que la *hull* consistance [4] et la *box* consistance [7] reposent sur l'application d'opérateurs de contractance extérieurs, qui éliminent ces valeurs en fonction d'une consistance donnée. Ces méhodes instancient le cadre de la consistance locale issu de l'intelligence artificielle [17]. De tels opéateurs bénéficient des propriétés de *contractance*, *complétude* et de *monotonie*.

Définition 2 (Opérateur Outer) *Soit* c *une contrainte n-aire et* ω *une consistance, un* opérateur de contraction extérieure *pour* c *est une fonction* $\mathsf{OCO}_c^{\omega} : \mathcal{I}^n \to \mathcal{I}^n$ *qui élimine des valeurs en fonction de la consistance* ω.

2.3 Approximations intérieures

Nous présentons ici des opérateurs fournissant une approximation *intérieure*, c'est-à-dire un sous-ensemble d'une relation ρ associée au système de contraintes. En conséquence, les boîtes résultant de l'application de ces opérateurs jouissent de la propriété de *correction*, c'est-à-dire que n'importe quel point appartenant à ces boîtes est solution du système de contraintes [6]. Nous considérons la définition suivante de l'approximation intérieure d'une relation :

Définition 3 (Opérateur Inner) *Soit une relation réelle n-aire* $\rho \subseteq \mathbb{R}^n$, *l'opérateur d'approximation intérieure* Inner$: \mathbb{R}^n \to \mathbb{R}^n$ *est défini par :*

$$\mathsf{Inner}(\rho) = \{\boldsymbol{r} \in \mathbb{R}^n \mid \mathsf{Outer}(\{\boldsymbol{r}\}) \subseteq \rho\}$$

Une telle approximation intérieure englobe tous les éléments dont la plus petite boîte englobante appartient à la relation.

2.3.1 Opérateur d'extension intérieure

Dans [10], Collavizza *et al.* proposent une approche qui permet de calculer, sur une dimension, une extension des domaines consistants d'une solution correcte d'un NCSP en utilisant des fonctions extremum monovariées pour évaluer les bornes minimum et maximum consistantes. Nous proposons une approche similaire basée sur un opérateur d'*extension intérieure* qui prend en entrée un pavé $\boldsymbol{B} \subseteq \rho_c$ consistant ainsi qu'une contrainte c et qui détermine un pavé $\boldsymbol{B'}$ tq $\boldsymbol{B'} \supseteq \boldsymbol{B}$ et dont toutes les valeurs satisfont la contrainte c.

Définition 4 (Opérateur d'extension intérieure) *Étant donné une contrainte n-aire c et sa relation sous-jacente ρ_c, un* opérateur d'extension intérieure *est une fonction $\mathsf{IEO}_c : \mathcal{I}^n \rightarrow \mathcal{I}^n$ vérifiant, pour toute boîte $\boldsymbol{B} \subseteq \rho_c$:*

$$\rho_c \supseteq \mathsf{IEO}_c(\boldsymbol{B}) \supseteq \boldsymbol{B}$$

Nous introduisons ensuite la notion d'opérateur optimal d'extension intérieure qui permet de déterminer la plus grande boîte (au sens de l'inclusion) dont tout point appartient à la relation ρ_c :

Définition 5 (Opérateur Optimal d'IEO) *Etant donné une contrainte n-aire c, un* opérateur optimal d'extension intérieure *pour c est une fonction $\mathsf{OptimalIEO}_c : \mathcal{I}^n \rightarrow \mathcal{I}^n$ vérifiant, pour tout $\boldsymbol{B} \supseteq \rho_c$:*

$$\nexists \boldsymbol{B'} \in \mathcal{I}^n | \boldsymbol{B'} \supset \mathsf{OptimalIEO}_c(\boldsymbol{B}) \wedge \boldsymbol{B'} \supseteq \rho_c$$

Devant l'impossibilité de déterminer numériquement une telle extension, due à la représentation des flottants en machine, nous proposons d'implémenter un opérateur approchant l'opérateur optimal. Cet opérateur d'extension intérieure est présenté par l'algorithme 1. Il est paramétré par une contrainte c, un domaine de recherche initial $\boldsymbol{B}$, un pavé consistant $\boldsymbol{P} \subseteq \rho_c$ et une précision ε. Une implémentation triviale consisterait à itérativement étendre la boîte $\boldsymbol{P}$ d'une valeur ε dans toutes les dimensions jusqu'à ce que le pavé $\boldsymbol{P}$ ne satisfasse plus la contrainte c (ou jusqu'à avoir dépassé l'espace de recherche $\boldsymbol{B}$) et à retenir l'avant dernière boîte calculée. Nous proposons ici une implémentation plus efficace basée une extension par dichotomie : une première itérative permet d'étendre la boîte consistante $\boldsymbol{P}$ selon la moitié du domaine disponible entre $\boldsymbol{P}$ et $\boldsymbol{B}$ pour chacune des dimensions jusqu'à trouver une boîte $\boldsymbol{P}$ partiellement consistante. Une deuxième étape réduit alors $\boldsymbol{P}$ de façon similaire jusqu'à trouver une boîte $\boldsymbol{P}$ certainement consistante par c. Le processus est récursivement appliqué jusqu'à obtention de la précision désirée. Dans l'algorithme Alg. 1, la notation $\boldsymbol{P}|_{v_i}$ signifie le domaine de la variable v_i dans la boîte $\boldsymbol{P}$, $\lfloor I$ signifie la borne gauche de l'intervalle I et $\lceil I$ sa borne droite.

L'approche peut être vue comme une extension à n dimensions de l'implémentation de l'opérateur de box-consistance [26], en garantissant cependant la correction plutôt que la complétude.

La figure 1 présente l'extension d'une boîte consistante $\boldsymbol{P}$, en fonction d'une contrainte c dans un espace de recherche limité à $\boldsymbol{B}$.

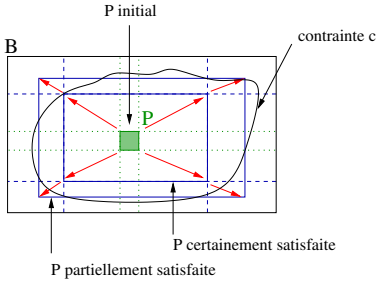

Fig. 1 – Extenstion (en deux dimensions) d'une boîte $\boldsymbol{P}$ en fonction d'une contrainte c dans une espace de recherche limité par une boîte $\boldsymbol{B}$.

2.3.2 Calcul d'extension intérieure pour un ensemble de contraintes $\{c_1, \ldots, c_n\}$.

Une fois l'extension intérieure présentée pour une contrainte c, nous considérons l'extension intérieure d'un ensemble $\{c_1, \ldots, c_n\}$ de contraintes dans une boîte $\boldsymbol{B}$, à partir d'une boîte initiallement consistante $\boldsymbol{P}$. Le principe est identique à celui formulé par [10] : pour toute contrainte c_i on détermine une extension intérieure avec l'opérateur IEO puis on les intersecte. L'algorithme obtenu AlgoIEA (*Inner Extending Algorithm*) est présenté par Alg. 2. Cet algorithme est paramétré par un ensemble $C = \{c1, \ldots, c_n\}$ de contraintes, une boîte $\boldsymbol{B}$ et une boîte consistante $\boldsymbol{P}$.

3 Résolution de problèmes Max-NCSP

Notre objectif est de proposer une méthode de résolution qui détermine une approximation intérieure des espaces de recherche satisfaisant le maximum de contraintes d'un CSP numérique. Nous proposons tout d'abord quelques définitions :

Définition 6 (Problème Max-NCSP) *Un problème Max-NCSP est défini par $\mathcal{S} = \langle C, V, \boldsymbol{D} \rangle$, où C est un ensemble de contraintes numériques, V*

ALG. 1 – Algorithme d'extension intérieure pour une contrainte c (AlgoIEO).

```
1 AlgoIEO_c(in: P ∈ I^n, B ∈ I^n, ε ∈ ℝ): out: K ∈ I^n
2  begin
3 if (smallestDifference(B, P) ≤ ε) then
4   return P
5 else
    % Boucle d'extension de P
6   while certainlySatisfied(P, c)
7     P' ← P
8     for each v_i ∈ c do
9           P|_{v_i} ← [ (⌊B|_{v_i} + ⌊P|_{v_i}) / 2 , (⌈B|_{v_i} + ⌈P|_{v_i}) / 2 ]
10    endforeach
11  end
    % Boucle de réduction de P
12  while not certainlySatisfied(P, c)
13    P'' ← P
14    for each v_i ∈ c do
15          P|_{v_i} ← [ (⌊P|_{v_i} + ⌊P'|_{v_i}) / 2 , (⌈P|_{v_i} + ⌈P'|_{v_i}) / 2 ]
15    endforeach
16  end
    % récursion
17  return AlgoIEO_c(P, P", ε)
18 end
19  end
```

ALG. 2 – Algorithme d'extension intérieure pour un ensemble C de n contraintes (AlgoIEA)

```
1 AlgoIEA(in: C; P ∈ I^n, B ∈ I^n, ε ∈ ℝ)
            out: K ∈ I^n
2 begin
3     return ⋂_{c_i∈C}^{n}  AlgoIEO_{c_i}(P, B, ε)
4 end
```

un ensemble de variables et $\boldsymbol{D}$ un produit Cartésien d'intervalles bornés par des nombres flottants.

Une solution $\boldsymbol{B} \subseteq \boldsymbol{D}$ du problème Max-NCSP $\mathcal{S}$ est une assignation des variables de V dans $\boldsymbol{D}$ telle que $\nexists \boldsymbol{B'} \subseteq \boldsymbol{D}$ *t.q.* $\mathsf{psatis}_C(\boldsymbol{B'}) > \mathsf{psatis}_C(\boldsymbol{B})$, avec $\mathsf{psatis}_C(\boldsymbol{B})$ qui détermine le cardinal des contraintes de C partiellement satisfaites pour la boîte $\boldsymbol{B}$ (partiellement satisfaite signifiant qu'il existe dans $\boldsymbol{B}$ des réels satisfaisant C et d'autres ne satisfaisant pas C).

Définition 7 (Approx. int. d'un Max-NCSP) *L'approximation intérieure d'un problème Max-NCSP $\mathcal{S} = \langle C, V, \boldsymbol{D}\rangle$ est composée de toutes les sous-régions $\boldsymbol{B} \subseteq \boldsymbol{D}$ telles que $\nexists \boldsymbol{B'} \subseteq \boldsymbol{D}$* t.q. $\mathsf{csatis}_C(\boldsymbol{B'}) > \mathsf{csatis}_C(\boldsymbol{B})$, *avec $\mathsf{csatis}_C(\boldsymbol{B})$ la fonction qui retourne le nombre de contraintes de C certainement satisfaites pour la boîte $\boldsymbol{B}$.*

Calculer une approximation intérieure d'un problème Max-NCSP consiste donc à fournir un ensemble de boîtes maximisant le nombre de contraintes certainement satisfaites du système. Etant donné un problème Max-NCSP $\langle C, \mathcal{V}, \boldsymbol{D}\rangle$, chaque zone de l'espace de recherche (c'est-à-dire un sous-ensemble de $\boldsymbol{D}$) peut être caractérisée par les ensembles disjoints de contraintes $\mathcal{C}, \mathcal{P}, \mathcal{N}$ tels que $\mathcal{C} \cup \mathcal{P} \cup \mathcal{N} = C$ et qui correspondent respectivement aux ensembles de contraintes $\mathcal{C}$ertainement satisfaites, $\mathcal{P}$ossiblement satisfaites et $\mathcal{N}$on satisfaites pour la boîte $\boldsymbol{B}$. Un exemple de boîtes certainement, possiblement et non satisfaites est donné dans la figure 2.

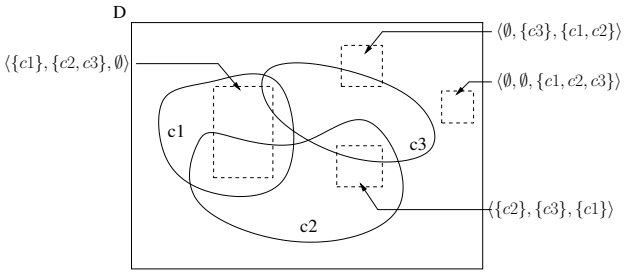

FIG. 2 – $\mathcal{CPN}$-caractérisation de différentes boîtes dans un pavé $\boldsymbol{D}$ en considérant trois contraintes $c1, c2, c3$.

Dans la suite de l'article, nous définissons trois opérateurs CS, PS et NS qui calculent respectivement les sous-ensembles $\mathcal{C}, \mathcal{P}$ et $\mathcal{N}$ d'une boîte $\boldsymbol{B}$ d'un problème Max-NCSP. Dans les trois définitions suivantes, nous considérons une boîte $\boldsymbol{B}$ de l'espace de recherche et un ensemble C de contraintes.

Définition 8 (Opérateur CS) *L'opérateur* $\mathsf{CS}(C, \boldsymbol{B})$ *calcule le sous-ensemble* C' *de contraintes de* C *tel que* $\forall c \in C', \boldsymbol{B} \subseteq \rho_c$.

Définition 9 (Opérateur PS) *L'opérateur* $\mathsf{PS}(C, \boldsymbol{B})$ *calcule le sous-ensemble* C' *de contraintes de* C *tel que* $\forall c \in C', \boldsymbol{B} \not\subseteq \rho_c \wedge \boldsymbol{B} \cap \rho_c \neq \emptyset$.

Définition 10 (Opérateur NS) *L'opérateur* $\mathsf{NS}(C, \boldsymbol{B})$ *calcule le sous-ensemble* C' *de contraintes de* C *tel que* $\forall c \in C', \boldsymbol{B} \cap \rho_c = \emptyset$.

Une fois ces trois opérateurs donnés, nous proposons la définition d'une boîte "$\mathcal{CPN}$", qui associe à une boîte $\boldsymbol{B}$, les trois ensembles de contraintes $\mathcal{C}, \mathcal{P}$ et $\mathcal{N}$.

Définition 11 (Boîte $\mathcal{CPN}$) *Une boîte* $\mathcal{CPN}$ *est un quadruplet* $\langle \boldsymbol{B}, \mathcal{C}, \mathcal{P}, \mathcal{N} \rangle$ *défini pour un ensemble de contraintes* C *et une boîte* $\boldsymbol{B}$ *tel que* $\mathcal{C} = \mathsf{CS}(C, \boldsymbol{B})$, $\mathcal{P} = \mathsf{PS}(C, \boldsymbol{B})$ *et* $\mathcal{N} = \mathsf{NS}(C, \boldsymbol{B})$.

3.1 Un premier algorithme naïf

Afin de déterminer une approximation intérieure des solutions d'un problème Max-NCSP, nous proposons une première approche basée sur l'*évaluation-bissection* à l'instar des travaux de Haroud et de Jaulin [16, 14] sur l'approximation intérieure d'un NCSP. L'étape d'évaluation consiste à déterminer la consistance de chacune des contraintes via les opérateurs CS, PS et NS pour une boîte donnée. L'exploration d'une branche de l'arbre ainsi produit par bissection-évaluation se termine dès que la boîte considérée ne possède plus de contraintes partiellement satisfaites ou est plus petite qu'un seuil donné. Les zones Max-NCSP sont alors formées par les noeuds possèdant le nombre de contraintes certainement satisfaites $\mathcal{C}$ le plus important. L'ensemble de l'espace est ainsi exploré.

Cette approche est représentée par l'algorithme IAMaxNCSP-SplitEval (*cf.* Alg.3). Il garantit la propriété de correction des solutions. Il est paramétré par un ensemble C de contraintes ainsi qu'une boîte $\boldsymbol{B}$ représentant l'espace de recherche initial et retourne la liste des pavés composant les meilleures approximations intérieures de ce Max-NCSP.

3.2 Une approche par extension intérieure

Afin d'obtenir ces résultats de manière plus efficace, nous proposons d'utiliser une approche de type *branch-and-bound* en maintenant à jour un optimum m sur le nombre de contraintes certainement satisfaites. À l'instar d'un processus d'optimisation classique, nous cherchons à élaguer au plus tôt les branches de recherche inintéressantes. Dans notre cas, ce sont

Alg. 3 – Algorithme naïf d'approximation intérieure pour un Max-NCSP.

```
1  IAMaxNCSP-SplitEval(in: C; B ∈ I^n)
       out: une liste ResultL de boîtes CPN
2  begin
3    ResultL ← ∅
4    L ← {⟨B, CS(C), PS(C), NS(C)⟩}
5    while not empty(L)
6      ⟨B, cs, ps, ns⟩ ← chooseOneFrom(L)
7      if (|ps| = 0) then
8        ResultL ← ResultL ∪ ⟨B, cs, ps, ns⟩
9      else
         % il reste des contraintes part. satisfaites
10       L ← L ∪ split(B)
11     end
12   endwhile
13   return best(ResultL)
14 end
```

les boîtes $\boldsymbol{B}$ telles que $|\mathsf{PS}(C, \boldsymbol{B})| + |\mathsf{CS}(C, \boldsymbol{B})| < m$). Nous améliorons ce processus classique sur trois points :

- nous proposons d'évaluer une boîte canonique $\boldsymbol{P}$ choisie dans $\boldsymbol{B}$ afin de mettre à jour plus rapidement l'optimum m. La boîte canonique satisfait possiblement plus de contraintes que $\boldsymbol{B}$ (étant canonique il y a des chances qu'elle possède moins de contraintes partiellement satisfaites). Cette heuristique permet d'élaguer les branches plus tôt dans le processus.
- nous proposons de déterminer une boîte $\boldsymbol{B'}$ par l'utilisation de l'opérateur d'extension intérieure à partir de la boîte $\boldsymbol{P}$ pour réduire le travail de recherche dans la boîte $\boldsymbol{B}$.
- enfin nous proposons l'emploi d'un opérateur d'approximation extérieur autour de la boîte $\boldsymbol{B'}$; celui-ci porte seulement sur les contraintes partiellement satisfaites. Si la propagation échoue (inconsistance), la boîte ne sera jamais meilleure que $|\mathsf{S}(C, \boldsymbol{B'})| + |\mathsf{PS}(C, \boldsymbol{B})|$. Si la propagation réduit partiellement les domaines, le pavé réduit est une zone prometteuse de recherche.

L'algorithme est présenté par Alg. 4. Considérons à un certain niveau du processus de résolution une boîte $\mathcal{CPN}$ $\langle \boldsymbol{B}, cs, ps, ns \rangle$ (ligne 6). En premier lieu, nous récupérons un point (*i.e.* une boîte canonique) par application d'une procédure chooseConfiguration qui retourne un point dans le pavé courant. Dans le cas où ce dernier est meilleur (satisfait certainement plus de contraintes que l'optimum courant), l'optimum est

ALG. 4 – Algorithme d'approximation intérieure pour un Max-NCSP.

```
1  IAMaxNCSP(in: C; B ∈ I^n)
       out: une liste ResultL de boîtes CPN
2  begin
3  maxSat ← 0
4  L ← {⟨B, CS(C), PS(C), NS(C)⟩}
5  while ¬empty(L)
6    ⟨B, cs, ps, ns⟩ ← chooseOneFrom(L)
7    if (|ps| = 0) then
8      ResultL ← ResultL ∪ ⟨B, cs, ps, ns⟩
9    else
       % recherche d'un canonique dans B
10     ⟨P, cs', ps', ns'⟩ ← chooseConfiguration(B)
11     if (|cs'| > maxSat) then
         % m.à.j. de l'optimum
12       maxSat ← |cs'|
13     end
       % branche intéressante ?
14     if (|cs' + ps'| > maxSat) then
15       B' ← AlgoIEA(cs', P, B, ε)
16       B'' ← propagation(B', ps')
17       L ← L ∪ split(B'')
18       L ← L ∪ {B'\B''}
19       L ← L ∪ {B\B'}
20     end
21   end
22 endwhile
23 return best(ResultL)
24 end
```

naturellement mis à jour. Nous pratiquons ensuite une extension intérieure (*i.e.* application de l'algorithme AlgoIEA) afin de maximiser la boîte. Une étape de propagation de contraintes en utilisant des opérateurs de contraction extérieurs est appliquée par la suite sur la boîte étendue.

Il est naturel de penser que la performance de cet algorithme dépend du moment de l'exécution où l'on détermine un des pavés qui satisfait maximalement le CSP. Plus celui-ci est trouvé tôt, plus les branches peuvent être élaguées rapidement. Dans la suite nous proposons d'instancier la méthode chooseConfiguration par une méthode de recherche locale afin de trouver au plus tôt l'optimum.

4 Une extension continue de la RL

Les méthodes de résolution incomplètes ont été appliquées avec succès à un grand nombre de CSPs discrets, voir [13] et plus récemment [8]. Parmi elles, les algorithmes de recherche locale sont basés sur un processus itératif d'amélioration d'une configuration initiale choisie aléatoirement. L'amélioration d'une configuration est relative à une notion de fonction de pénalité f (aussi appelée fonction de coût ou fonction d'erreur). Cette dernière représente le plus souvent dans le cadre des CSPs, le nombre de contraintes violées par la configuration courante (selon l'idée du Min-Conflict [19]).

Le principe de ces algorithmes est de débuter avec une configuration initiale aléatoire, d'explorer un voisinage de cette configuration et de modifier la configuration courante sous certaines conditions : combinaison d'amélioration de la pénalité et d'une métaheuristique. On répète ces étapes jusqu'à satisfaction d'une condition de terminaison (une solution a été trouvée ou un nombre maximum d'itérations a été atteint). La partie métaheuristique consiste à guider la recherche à un plus haut niveau que l'itération élémentaire, ceci pour éviter de boucler à court terme, et pour garantir certaines propriétés de l'algorithme par rapport à l'espace de recherche (intensification, diversification). L'une des métaheuristiques les plus efficaces et connues est certainement la recherche taboue proposée par Glover [11] et séparément par Hansen [23].

La recherche locale (et les métaheuristiques en général) est souvent guidée avec deux notions complémentaires. La diversification consiste à garantir que la recherche explore largement l'espace de recherche, sans rester confinée à un sous-espace précis, ce qui peut-être réalisé simplement avec des réinitialisations aléatoires. Au contraire, l'intensification consiste à forcer la recherche dans

certaines zones supposées prometteuses.

Les applications des techniques métaheuristiques aux CSPs continus restent encore assez rares. Plusieurs ont été proposées à partir d'approches populationnistes, comme par exemple [18]. Les travaux de Chelouah et Siarry ont étendu de nombreuses méta-heuristiques au cas continu, avec notamment la recherche taboue avec ECTS [9]. Une difficulté est de tenir compte de domaines bien plus grands que dans le cas discret. ECTS utilise une métrique sur ces domaines pour définir des rectangles concentriques autour de la configuration courante, dans lesquels on tire aléatoirement les configurations du voisinage. Une autre recherche taboue continue est proposée par Battiti et Tecchiolli [3] dans un algorithme hybride. Ici, le continu est traité comme une grille de points recouvrant l'espace de recherche $\mathbb{R}^n$. Anglada *et al.* [2] proposent eux aussi une adaptation au continu d'un algorithme proche de la recherche taboue, la recherche adaptative. Le voisinage dépend de la fonction à optimiser, en coupant par exemple sur une variable donnée les sous-ensembles de son domaine améliorant la fonction.

Dans ces travaux, les configurations étudiées sont comme dans le cas discret des instanciations de valeurs (continues) aux variables, donc des points de $\mathbb{R}^n$. Cela amène à redéfinir notamment le voisinage et la partie heuristique des algorithmes, pour tenir compte de la grande taille des domaines explorés ($\mathbb{R}$ discrétisé). Notre approche est orthogonale : pour les besoins de notre application, nous devons considérer non pas des points de $\mathbb{R}^n$ mais des intervalles de $\mathcal{I}^n$. Nous avons donc d'abord centré l'algorithme sur la gestion de ces configurations-pavés et la nature continue des contraintes. La partie métaheuristique ou recherche taboue de notre algorithme reste assez simple pour l'instant, ce que nous justifierons ci-dessous. Cette méthode se rapproche des travaux de Barichard et Hao [25], qui proposent PICPA, une méthode hybride dont la partie méta-heuristique est un algorithme génétique. Les individus sont des boîtes sélectionnées selon un critère multi-objectif. Dans notre algorithme, les configurations sont des boîtes sélectionnées selon un critère Min-Conflict.

4.1 Configurations d'intervalles

Dans le cas discret, on considère généralement une méthode de recherche locale comme décrite par le triplet :

- fonction de voisinage, qui fait correspondre un ensemble de voisins à chaque configuration de l'espace de recherche ;
- fonction de pénalité à optimiser ;
- heuristique permettant d'éviter le bouclage de l'algorithme.

Cependant les algorithmes de recherche locale utilisent également des opérateurs qui ne sont jamais définis, voire même jamais identifiés car leur instanciation est évidente pour le cas discret. En premier lieu, la notion de configuration est immédiate, celle-ci étant définie comme une affectation de valeurs aux variables discrètes. De ceci, découle le reste de manière évidente : deux configurations sont comparées par rapport à l'égalité des valeurs de leurs variables. Une solution est trivialement identifiée comme ayant une pénalité nulle. Ces opérations évidentes sur le discret ne le sont plus sur le continu.

Comme dit plus haut, notre application nécessite des variables continues au sens le plus large, dont leurs domaines sont représentés comme des intervalles à bornes flottantes. Pour envisager une technique de recherche locale sur les intervalles, il faut d'abord redéfinir plusieurs opérations élémentaires.

En premier lieu, il faut définir l'ensemble S des configurations possibles. Une possibilité est de choisir $\mathbb{F}^n$ comme dans le cas discret. Mais cette solution ne tirerait pas avantage des propriétés de continuité des domaines présentées en section 2.1. De plus, le calcul des pénalités serait hasardeux tout comme la vérification des contraintes pour les méthodes complètes : erreurs d'arrondi, probabilité très faible de satisfaire une contrainte d'égalité ($\mathbb{F}$ étant un sous-ensemble dénombrable de $\mathbb{R}$). De plus, dans le cas des inégalités, nous voulons calculer un pavé solution (c'est-à-dire une approximation intérieure de l'ensemble solution). Nous définissons donc naturellement S comme étant l'ensemble des intervalles $\mathcal{I}^n$.

4.2 Choix des opérateurs de RL

A partir de ce choix pour S, il nous faut définir de manière cohérente les notions de fonction de voisinage et d'exploration, de fonction de pénalité et de comparaison de configurations.

4.2.1 Fonction de voisinage

La fonction de voisinage en recherche locale doit retourner, partant de la configuration courante, les configurations potentiellement sélectionnables pour un mouvement. On considère des configurations voisines dans le sens à la fois géométrique (proches de la configuration courante) et calculatoire (le voisinage doit être obtenu simplement).

Nous définissons le voisinage d'une configuration p à partir d'une boîte $\boldsymbol{B}_p$ centrée sur la configuration courante. Par principe, il faudrait calculer un nombre

très important de boîtes (ordre de grandeur de l'ensemble des parties d'un ensemble de $\boldsymbol{B}_p \cap \mathbb{F}$) pour couvrir le voisinage. Aussi nous introduisons un nouvel opérateur génèrant aléatoirement nb_{neigh} nouvelles configurations à l'intérieur de la boîte $\boldsymbol{B}_p$, qui seront le voisinage à explorer

Notre *fonction de voisinage* sera parametrée par le nombre de voisins générés à chaque étape (nb_{neigh}). On peut rapprocher notre définition de *fonction de voisinage* à l'idée de *voisinage variable* ou VNS (*Variable Neighbourhood Search*) présentée par Hansen et Mladenovi'c dans [20].

Il est à noter que l'explosion combinatoire du nombre configurations-pavés, qui nous gêne pour explorer le voisinage, nous protège du bouclage à court terme et nous évite de définir une liste taboue au sens strict. Dans le cas discret, une configuration a de grandes chances d'être choisie plusieurs fois dans un nombre relativement restreint d'itérations (taille et reversibilité du voisinage). Dans notre cas, la probabilité de recalculer deux fois la même configuration est négligeable. Bien sûr, ceci n'empêche pas l'immobilisation de la recherche dans une zone de minimum local. Les questions d'intensification, diversification et métaheuristique pour les intervalles restent à développer plus en détail. Dans l'état actuel de l'algorithme, nous exécutons des redémarrages aléatoires pour assurer la diversification.

4.2.2 Fonction de pénalité

Les contraintes étant définies sur les réels, par des opérateurs arithmétiques usuels, ce sont des fonctions de $\mathbb{R}$ dans $\{0, 1\}$. Ici nous voulons optimiser selon la satisfaction des contraintes, en tenant compte des différents cas possibles pour les contraintes continues, où la notion de solution est découpée en approximations intérieure et extérieure. Nous procédons pour cela en plusieurs étapes.

Nous avons un seul type de contraintes, des inégalités du type $f(V) \leq g(V)$, où f et g sont des fonctions réelles sur un sous-ensemble des variables. Dans le cas d'un cercle centré en 0 dans le plan, par exemple, on aura $f(x, y) = \sqrt{x^2 + y^2}$ et g constante. Il faut tenir compte de l'évaluation des pénalités sur des boîtes et non des points. Ici nous bénéficions de la puissance de l'extension aux intervalles (2.1). On sait calculer l'intervalle image de la configuration pour f et g, via les extensions F et G. Cet intervalle représente aussi précisement qu'il est possible les valeurs prises par la configuration courante. Enfin, nous comptons la pénalité à la façon Min-Conflict : pour la contrainte considérée, la position relative des boîtes calculées permet de savoir si la contrainte est entièrement satisfaite pour cette configuration, partiellement satisfaite ou entièrement non-satisfaite. Le principe est le même que le Min-Conflict discret, il s'agit intuitivement de mesurer si la contrainte est satisfaite ou non, mais de manière etendue au cas des intervalles. On obtient pour chaque contrainte un triplet de valeurs comme illustré par la figure 2. En sommant ces valeurs, on obtient le nombre de contraintes satisfaites (certainement, partiellement, ou non). Nous retenons dans la version actuelle de l'algorithme le nombre de contraintes certainement satisfaites comme fonction de pénalité.

4.2.3 Algorithme de RL continue

Nous donnons ici le pseudo-code de notre algorithme. Il nécessite quatre paramètres :

1. le nombre de *réinitialisations aléatoires* : maxTries ;
2. le nombre d'itérations associées à chaque point de départ : maxSteps ;
3. le nombre de voisins nb_{neigh} générés à chaque itération ;

ALG. 5 – Recherche Locale continue.

```
1 CLS(in: C; P ∈ I^n;
        in: maxTries,maxSteps,nb_neigh ∈ ℤ ;)
        out: B ∈ I^n
2 begin
3   best ← initialConfiguration(C, P)
4   current ← best
4   nbTries ← 0
    % Boucle de Diversification
5   while nbTries < maxTries
6     nbSteps ← 0
      % Boucle d'Intensification
7     while nbSteps < maxSteps
8       neigh ← genNeighbors(nb_neigh, current, C)
9       current ← getBestNeighbor(neigh, C)
10      if (cost(current) ≺_I cost(best)) then
11        best ← current
12      end
13      nbSteps + +
14    endwhile
15    nbTries + +
16  endwhile
17  return best
18 end
```

Notre algorithme est présenté dans ALG. 5. Ce dernier prend en entrée un ensemble de contraintes

$C = \{c_1, \ldots, c_n\}$, une boîte de recherche $\boldsymbol{P}$ et les trois paramètres de la recherche locale.

5 Résultats

Nous présentons ci-dessous les résultats obtenus, en comparant les temps d'execution des algorithmes IAMaxNCSP-SplitEval, IAMaxNCSP-Mid et IAMaxNCSP-CLS pour le même ensemble de problèmes. L'algorithme IAMaxCSP-Mid représente l'algorithme IAMaxNCSP dans lequel la méthode chooseConfiguration() retourne simplement le milieu du pavé. L'algorithme IAMaxNCSP-CLS représente IAMaxNCSP dans lequel chooseConfiguration() fait appel à la recherche locale sur le continu (CLS). Les problèmes jouets Circle-n définissent un ensemble de n disques dont les rayons et centres sont définis aléatoirement. L'espace de recherche est restreint à l'intervalle $[-3, 3]^2$. Les problèmes Sphere-n définissent de façon identique des spheres aléatoirement réparties (dimension $[-3, 3]^2$). La recherche locale est ici paramétrée par maxTries=10, maxSteps=10, nb_{neigh}=10.

Les résultats obtenus montrent de façon claire que l'approche avec recherche locale n'arrive pas à sérieusement concurrencer l'approche par point intérieur, malgré les différents paramètrages applicables sur la recherche locale (nombre de voisins, d'étapes et de random restart). Si, dans l'ensemble des cas, la recherche locale trouve l'optimum (le maximum de contraintes certainement satisfaites) plus tôt que la méthode point milieu (et souvent dès la première étape), en revanche, cette recherche locale perd le reste du temps en évaluations dans des zones très peu prometteuses (typiquement dans tous les pavés sur les frontières des relations). Le gain fourni aux premières étapes est alors perdu par la suite. Quelques tentatives de méthodes adaptatives ont été proposées (favoriser la recherche locale en début de processus, puis utiliser la méthode du point milieu), mais ne sont pas performantes de la même façon sur les différents benchmarks.

Tab. 1 – Résultats des approches SplitEval, Mid et CLS en secondes sur un Pentium M 1.7 GHz, 512Mo.

Bench. IAMaxCSP	-SplitEval time	-Mid time	-CLS time
$\varepsilon = 0.1$			
Circle5 (2-sat)	0.49	0.09	0.22
Circle25 (9-sat)	0.96	0.13	0.05
Circle100 (21-sat)	3.99	1.3	0.9
Sphere10 (3-sat)	10.19	1.21	1.21
Sphere100 (9-sat)	157.74	9.29	2.39
$\varepsilon = 0.01$			
Circle5 (2-sat)	3.87	1.9	1.8
Circle25 (9-sat)	16.14	0.54	0.51
Circle100 (21-sat)	133.04	1.02	1.62
Sphere10 (3-sat)	107.76	75.6	77.31
Sphere100 (9-sat)	$>$ 10mn	3.64	3.63
$\varepsilon = 0.001$			
Circle5 (2-sat)	24.87	3.92	7.56
Circle25 (9-sat)	127.64	2.0	1.72
Circle100 (21-sat)	$>$ 10mn	4.31	3.80
Sphere100 (9-sat)	$>$ 10mn	71.53	72.52

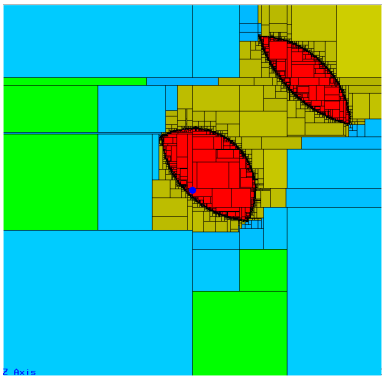

Fig. 3 – Application de notre approche sur Circle5 ($\varepsilon = 0.01$). Les zones rouges représentent les pavés satisfaisant maximisant le CSP numérique (deux contraintes sur cinq).

6 Conclusion

Cet article a présenté une méthode pour résoudre les problèmes Max-NCSP en introduisant un opérateur d'extension intérieure couplé avec des opérateurs classiques de contraction. Des résultats sont présentés sur des exemples simples. Un couplage avec une méthode de recherche locale a été proposé sans toutefois obtenir les résultats escomptés. Un cadre générique a cependant été défini pour la recherche locale continue, de manière orthogonale aux méthodes

existantes où les configurations sont des points. Notre cadre est centré sur les intervalles, seule manière à répondre aux questions classiques d'approximations intérieures et extérieures. La contrepartie est la difficulté à définir la partie méta-heuristique, qui doit être développée.

Références

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press Inc., New York, USA, 1983.

[2] Alexis Anglada, Philippe Codognet, and Laurent Zimmer. An adaptive search for the NSCSPs. In *CSCLP 2004*, Lausanne, 2004.

[3] R. Battiti and G. Tecchiolli. The continuous reactive tabu search : blending combinatorial optimization and stochastic search for global optimization. *Annals of Operations Research*, 62, 1996.

[4] F. Benhamou. Interval constraint logic programming. In Andreas Podelski, editor, *Constraint programming : basics and trends*, volume 910 of *LNCS*, pages 1–21. Springer-Verlag, 1995.

[5] F. Benhamou and F. Goualard. Universally quantified interval constraints. In *Procs. of CP'2000*, pages 67–82, 2000.

[6] Frédéric Benhamou, Frédéric Goualard, Éric Languénou, and Marc Christie. Interval constraint solving for camera control and motion planning. *ACM Transactions on Computational Logic*, 5(4), October 2004.

[7] Frédéric Benhamou, David McAllester, and Pascal Van Hentenryck. CLP(Intervals) revisited. In *Procs. of the International Symposium on Logic Programming (ILPS'94)*, pages 124–138, Ithaca, NY, November 1994. MIT Press.

[8] C. Blum and A. Roli. Metaheuristics in combinatorial optimization : Overview and conceptual comparison. *ACM Computing Surveys*, 35 :268–308, 2003.

[9] R. Chelouah and P. Siarry. Tabu search applied to global optimization. *European Journal on Operational Research*, 2000.

[10] Hélène Collavizza, François Delobel, and Michel Rueher. Extending consistent domains of numeric CSP. In *Procs. of the 16th IJCAI*, volume 1, pages 406–411, Stockholm, Sweden, July 1999.

[11] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.

[12] Eldon Robert Hansen. *Global Optimization Using Interval Analysis*. Pure and Applied Mathematics. Marcel Dekker Inc., 1992.

[13] Jin-Kao Hao, Philippe Galinier, and Michel Habib. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Journal of Heuristics*, 1998.

[14] D. Haroud and B. Faltings. Global consistency for continuous constraints. *Lecture Notes in Computer Science*, 874 :40–50, 1994.

[15] L. Jaulin and E. Walter. Guaranteed tuning, with application to robust control and motion planning. *Automatica*, 32(8) :1217–1221, 1996.

[16] Luc Jaulin and Éric Walter. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, 29(4) :1053–1064, 1993.

[17] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 1(8) :99–118, 1977.

[18] Z. Michalewicz. Genetic Algorithms, Numerical Optimization and Constraints. *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 151–158, 1995.

[19] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58 :161–205, 1992.

[20] N. Mladenović and P. Hansen. Variable Neighborhood Search. *Comps. in Opns. Res.*, 24 :1097–1100, 1997.

[21] R. E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1966.

[22] Arnold Neumaier. *Interval methods for systems of equations*, volume 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge Unversity Press, 1990.

[23] Hansen P. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, 1986.

[24] Jamila Sam-Haroud and Boi V. Faltings. Consistency techniques for continuous constraints. *Constraints*, 1 :85–118, 1996.

[25] Barichard V. and Jin-Kao Hao. A population and interval constraint propagation algorithm. *LNCS*, 2632 :88–101, 2003.

[26] Pascal Van Hentenryck, Laurent Michel, and Yves Deville. *Numerica : A Modeling Language for Global Optimization*. The MIT Press, 1997.