Arbres d'itérations chaotiques pour décrire la résolution des CSPs

Julien Arsouze, Gérard Ferrand, Arnaud Lallouet
Université d'Orléans — LIFO
BP 6759 — 45067 Orléans cedex 2
{prénom.nom}@lifo.univ-orleans.fr

Résumé

Ces travaux sont motivés par la mise au point des programmes avec contraintes, en particulier par l'obtention des traces de calculs. Le formalisme présenté ici permet de décrire le mécanisme de résolution des solveurs. La recherche des solutions d'un problème de satisfaction de contraintes consiste en un entrelacement de pas de réduction et de pas de recherche qui sont vus comme des étapes d'ajout de contraintes. Pour modéliser cela, on définit des arbres de recherche qui ont des propriétés de correction, complétude, indépendance par rapport à la règle de choix analogue à celles des arbres SLD en programmation logique.

This works are motivated by the debugging of programs with constraints, in particular to obtain some computation traces . The formalism presented here aims at describing solvers mecanisms. The search for solutions of a constraint satisfaction problem consists in a mix of reduction and search steps which are seen as an addition of constraints. In order to modelize this technic, search trees are defined. They have properties of correctness, completness and independance of the choice rule. This properties are similar to SLD trees ones in logic progamming.

mots clés : CSP, consistance, itération chaotique, équité, labeling, arbre de recherche

Keywords : CSP, consistency, chaotic iteration, fairness, choice set, labeling, search tree

1 Introduction

Un Problème de Satisfaction de Contraintes (CSP) est par définition un ensemble de contraintes. Chaque variable d'une contrainte a un domaine associé qui contient les valeurs possibles de la variable. Une contrainte est définie par une relation entre différentes variables. On s'intéresse ici uniquement aux contraintes sur les domaines finis. Résoudre un CSP consiste à trouver ses solutions, c'est à dire trouver les différentes affectations possibles pour chaque variable qui satisfassent les contraintes. Une énumération de toutes les combinaisons possibles déboucherait sur une explosion combinatoire. C'est pourquoi les méthodes de propagation de contraintes sont utilisées. Le principe est d'éliminer les valeurs qui ne peuvent satisfaire les contraintes. Cette réduction des domaines est effectuée par des algorithmes qui appliquent des propriétés de consistance [14]. Les itérations chaotiques d'opérateurs corrects, monotones et contractants permettent de modéliser la sémantique opérationelle de ces solveurs de contraintes [2], [9].

Dans cette formalisation classique, à chaque pas d'itération, n'importe quel opérateur est choisi. Il en suit que les itérations chaotiques sont par nature infinie. Dans ce papier, l'ensemble des contraintes que doivent satisfaire les variables est représenté abstraitement par un ensemble γ d'opérateurs de réduction "choisissables" appelé *ensemble de choix*. En plus de modéliser les algorithmes existant, cela permet de detecter la terminaison d'une itération chaotique (quand $\gamma = \emptyset$). Mais, le point fixe d'un ensemble d'opérateurs de réduction vérifie seulement une consistance, ce qui est en général insuffisant pour trouver les solutions. C'est pourquoi des algorithmes de recherche (souvent décrits de façon indépendante dans un langage hôte) sont utilisés en plus des méthodes de propagation. Ceci est pris en compte dans notre formalisme car un ensemble d'opérateurs de réduction (incorrects par rapport aux solutions) associé à un ensemble de contraintes permet de se focaliser sur des parties de l'espace de recherche. Ce procédé de recherche est organisé par des arbres de recherche dont les propriétés de correction, de complétude et d'indépendance rappellent celles des arbres SLD de la programmation logique.

Ces travaux sont motivés par la mise au point des programmes avec contraintes, en particulier par l'obtention des traces de calculs [10], [7] en utilisant le formalisme présenté ici qui permet de décrire le mécanisme de résolution des solveurs.

Le plan du papier est le suivant. La partie 2 rappelle les résultats de correction et de complétude pour les itérations chaotiques. La partie 3 présente un modèle déclaratif du choix des opérateurs basé sur des ensembles de choix qui restreignent le choix à chaque pas de calcul. On définit alors les γ-itérations chaotiques. Grâce aux schémas de calcul qui représentent la suite des états de calcul d'un solveur, les mécanismes de recherche sont pris en compte et les arbres de recherche qui organisent la recherche et la réduction peuvent être définis. Ces contributions sont présentées dans la partie 4, où un exemple de recherche, le labeling, est également décrit en détails.

2 problèmes de satisfaction de contraintes

Ce papier traite uniquement des méthodes de réduction de domaines et de recherche pour la résolution des CSPs. On se réfèrera à Tsang [16] pour les autres méthodes de résolution.

Soient A un ensemble et I un ensemble fini d'indices. $(a_i)_{i\in I}$ avec $a_i \in A$ représente une famille d'éléments de A indexée par I. Si A est une famille d'ensembles $(A_i)_{i\in I}$, $\Pi A = \Pi_{i\in I} A_i = \{(a_i)_{i\in I} \mid \forall i \in I, a_i \in A_i\}$ s'appelle le produit cartésien *indexé* de la famille A. Les éléments de ΠA sont des *n-uplets*. Si tous les ensembles A_i sont ordonnés, ΠA est ordonné par le produit des ordres de tous les A_i. Pour $J \subseteq I$ et une famille a sur I, $a|_J = (a_i)_{i\in J}$ est la restriction de a à l'ensemble d'indices J. $a|_J$ est un élément du produit cartésien indexé $\Pi(A|_J)$. Enfin, $\mathcal{P}(A)$ représente l'ensemble des parties de A.

Soient V un ensemble fini de variables, C un ensemble fini de contraintes. La fonction $var : C \longrightarrow \mathcal{P}(V)$ associe à une contrainte l'ensemble de ses variables. Soit $D = (D_x)_{x\in V}$ une famille de domaines finis non vide. Une famille d'ensembles de n-uplets $ext \in \Pi_{c\in C}\mathcal{P}(\Pi_{i\in var(c)}(D_i))$ définit l'*extension* des contraintes. Pour chaque contrainte c, $ext_c \subseteq \Pi(D|_{var(c)})$ s'appelle l'extension de c, c'est à dire la définition, ou l'ensemble solution de la contrainte c.

Définition 1 (CSP) *Un CSP est défini par un sous-ensemble $\mathcal{C}$ de C.*

La résolution des CSPs est surtout un problème de réduction des extensions des contraintes [15, 2]. Ici, on s'appuie sur le "*value constraint* subscheme" de Van Emden [17]. A chaque variable x de V est associée une contrainte unaire d_x qui est appelée domaine de x. Il est le plus utile en pratique. C'est en particulier la base du système du solveur de contraintes de CHIP [18], du système Gnu-Prolog [8] et de bien d'autres. L'utilisation de ce cadre fait que l'*espace de recherche* est $\mathcal{SS} = \Pi_{x\in V}\mathcal{P}(D_x)$. Un élément de $\mathcal{SS}$ est un *état de recherche*. $\mathcal{SS}$ est un treillis qui est naturellement ordonné par $\subseteq$, le produit des sous-ordres de chaque domaine des variables.

Soit $I = \mathbb{N}$ ou $I = [0\ldots n]$ $n \in \mathbb{N}$ un ensemble d'indices et $I^+ = I - \{0\}$. Pour toute suite décroissante $e = (e_i)_{i\in I}$, où $e_i \in \mathcal{SS}$, la limite notée $\lim(e) = \bigcap_{i\in I} e_i$ existe. Un état de recherche d est dit *singletonique* si $\forall x \in V$, d_x est un singleton. Une famille $u = (u_x)_{x\in V} \in \Pi(D)$ est appelée *n-uplet global*. L'ensemble de tous les n-uplets globaux est noté $\mathcal{GT}$. Les solutions d'un CSP $\mathcal{C}$ sont les n-uplets globaux qui satisfont toutes les contraintes. Une bijection naturelle existe entre les n-uplets globaux et les états de recherche singletoniques. L'ensemble des solutions d'un CSP $\mathcal{C}$ est donc l'ensemble $Sol(\mathcal{C}) = \{u \in \mathcal{GT} \mid \forall c \in S, u|_{var(c)} \in ext_c\}$.

La recherche des solutions d'un CSP utilise deux techniques : la recherche et la réduction de domaines qui supprime les valeurs inconsistantes des domaines des variables. Cette réduction est faite par un algorithme qui permet d'obtenir une propriété appelée *consistance*.

Exemple 2 (consistance locale) *Dans les CSPs binaires, une contrainte $c(x, y)$ est arc-consistante si pour toute valeur v_x du domaine de x, il existe une valeur v_y dans le domaine de y telle que $c(v_x, v_y)$ est vérifiée et réciproquement pour toute valeur du domaine de y. Un CSP est arc-consistant si toutes ses contraintes sont arc-consistantes. Beaucoup d'autres consistances existent dans la littérature comme entre autres l'hyper arc-consistance, la path-consistance, la k-consistance et l'arc consistance directionnelle, toute étant décrites avec les références exactes dans [16].*

Ces consistances locales peuvent être modélisées par le plus grand point fixe d'un ensemble d'opérateurs correctement définis. Cette technique a été introduite par Montanari et Rossi [15] puis étendue par [2]. Ces opérateurs sont définis à partir de règles qui réduisent le domaine d'une seule variable à partir des extensions d'autres contraintes :

Définition 3 (règle de consistance locale) *Soient $U \subseteq V$ et $x \in V$. Une* règle de consistance locale *est une fonction monotone $r : \Pi_{z\in U}\mathcal{P}(D_z) \longrightarrow \mathcal{P}(D_x)$.*

Chaque règle de consistance locale r est étendue pour devenir un *opérateur de réduction* $reduc_r$ sur $\mathcal{SS}$.

Définition 4 (opérateur de réduction) *Soient $r : \Pi_{z\in U}\mathcal{P}(D_z) \longrightarrow \mathcal{P}(D_x)$ une règle de consistance locale et $d \in \mathcal{SS}$. L'opérateur de réduction $reduc_r : \mathcal{SS} \longrightarrow \mathcal{SS}$ associé à r est défini par :*

- $\forall v \in V,\ v \neq x \Longrightarrow reduc_r(d)_v = d_v$
- $reduc_r(d)_x = r(d|_U) \cap d_x$

Exemple 5 (Gnu-Prolog) *Le système* Gnu-Prolog *[8] implante de telles régles pour la résolution de contraintes dans les domaines finis. Ces régles de réduction de domaine respectent une syntaxe particulière appelée "X in r" où X représente le domaine d'une variable qui doit être réduit et r est une expression dans laquelle se trouve le domaine d'autres variables. La sémantique opérationelle d'une telle régle est $d_X := d_X \cap eval(r)$.*

Une contrainte n-aire est compilée en régles "X in r", chacune d'elle définissant un opérateur. Par exemple, la contrainte $X < Y$ est compilée en "X in $0..min(Y) - 1$" et "Y in $max(X) + 1..infinity$". La composition des ces expressions permet de modéliser différentes consistances tel que l'(hyper)arc-consistance ou l'arc-consistance partielle.

En plus d'être monotones et contractant, tous les opérateurs de réductions sont supposés corrects.

Définition 6 (opérateur de réduction correct) *Un opérateur de réduction $reduc_r$ est* correct *si pour tout $d \in \mathcal{SS}$ et pour toute solution s d'un CSP, $s \in \Pi d$ implique $s \in \Pi reduc_r(d)$.*

On est donc assuré que l'application d'un opérateur de réduction ne supprime pas de solutions. Soit $\mathcal{F}$ un ensemble d'opérateurs de réduction. Les états de recherche d pour lesquels aucun opérateur de $\mathcal{F}$ n'a d'effet, c'est à dire $\forall f \in \mathcal{F}, f(d) = d$, sont appelés *états stables*.

Soient s une solution d'un CSP et $\mathcal{F}$ un ensemble d'opérateurs de reduction corrects alors s est stable par $\mathcal{F}$. Dans ce papier, la réciproque est aussi supposée, c'est à dire qu'un état singletonique est une solution du CSP si et seulement si il est stable par $\mathcal{F}$.

Soit $Cl_{\mathcal{F}}(d) = max\{d' \in SS \mid d' \subseteq d$ et d' stable par $\mathcal{F}\}$ la clôture de d par $\mathcal{F}$. L'opérateur $Cl_{\mathcal{F}}$ est l'*opérateur de clôture* associé à $\mathcal{F}$. Pour calculer la clôture d'un état d par un ensemble $\mathcal{F}$ d'opérateurs de réduction, on peut considérer deux sortes d'itérations. La première, l'*itération standard* consiste à itérer l'opérateur obtenu en combinant tous les opérateurs de $\mathcal{F}$. Soit $F_{\mathcal{F}}$ l'opérateur défini par : $\forall d \in \mathcal{SS},\ F_{\mathcal{F}}(d) = \bigcap_{f \in \mathcal{F}} f(d)$.

Définition 7 (iteration standard) *L'*itération standard *de $\mathcal{F}$ à partir de $d \in \mathcal{SS}$ est la suite $a = (a_i)_{i \in \mathbb{N}}$ d'éléments de $\mathcal{SS}$ tel que $a_0 = d$ et $\forall i \in \mathbb{N}^+,\ a_i = F_{\mathcal{F}}(a_{i-1})$.*

Pour obtenir l'étape suivante, tous les opérateurs doivent être appliqués. C'est pourquoi les itérations chaotiques sont souvent préférées. Elles ont été introduites par Chazan et Miranker [5] pour l'algébre linéaire. Elles ont été utilisées comme technique générale pour accélérer le calcul de points fixes par Cousot et Cousot [6] et dans beaucoup d'autres domaines avant de servir de fondation pour la propagation de contraintes ([9], [2], [17]). On peut noter que Montanari et Rossi [15] ont aussi établi le résultat principal mais pour des opérateurs plus spécifiques et que l'idée de calculer le plus grand point fixe est déjà présente chez Benhamou [4].

Définition 8 (itération chaotique) *Une* itération chaotique *de $\mathcal{F}$ à partir de $d \in \mathcal{SS}$ est une suite $b = (b_i)_{i \in I}$ d'éléments de $\mathcal{SS}$ avec $b_0 = d$ et $\forall i \in I^+,\ \exists f \in \mathcal{F}$ tel que $b_i = f(b_{i-1})$.*

Donc, à chaque pas, seulement un opérateur de réduction est appliqué et chaque application tire avantage des réductions faites par les pas précédents. Les itérations chaotiques dont la limite est stable sont dites *complètes.* Pour cette classe d'itération chaotique, la limite est la clôture $Cl_{\mathcal{F}}(d)$.

Le résultat d'une itération chaotique dépend de la stratégie utilisée pour l'ordre d'application des opérateurs. Le lemme suivant établit qu'une itération chaotique fournit une approximation supérieure de l'itération standard.

Lemme 9 *Soient $(a_i)_{i \in \mathbb{N}}$ l'itération standard de $F_{\mathcal{F}}$ and $(b_i)_{i \in I}$ une itération chaotique, chacune d'elle partant de $d \in \mathcal{SS}$. Alors $Cl_{\mathcal{F}}(d) \subseteq lim(a) \subseteq lim(b)$.*

Les itérations chaotiques complètes ont leur limite qui est égale à la limite de l'itération standard. Toutes deux sont égales à la clôture de leur état de départ d par $\mathcal{F}$.

Corollaire 10 (Correction) *De plus, si l'itération chaotique b est complète (c'est à dire $lim(b)$ stable), alors $lim(b) = lim(a) = Cl_{\mathcal{F}}(d)$.*

Preuve Tout d'abord, $Cl_{\mathcal{F}}(d) \subseteq b_0$. Donc, supposons que $Cl_{\mathcal{F}}(d) \subseteq b_i$. $\exists f \in \mathcal{F}$ $b_{i+1} = f(b_i)$. Alors $f(Cl_{\mathcal{F}}(d)) \subseteq f(b_i)$ et finalement $Cl_{\mathcal{F}}(d) \subseteq b_{i+1}$.

L'inclusion inverse est vérifiée car $Cl_{\mathcal{F}}(d)$ est le plus grand état stable par $\mathcal{F}$. Donc $lim(b) \subseteq Cl_{\mathcal{F}}(d)$.

La complétude d'une itération chaotique est donc équivalente à la conjonction des deux conditions suivantes : $lim(a)$ stable et $lim(b) = lim(a)$. Pour assurer $lim(b) = lim(a)$, c'est à dire $lim(b) \subseteq lim(a)$, l'équité de la stratégie d'ordonnancement des opérateurs de réduction est une condition suffisante (voir [11] pour une présentation générale de l'équité). Pour les itérations chaotiques infinies, l'équité donne une condition sur la stratégie d'ordonnacement. Intuitivement, cela consiste à s'assurer que l'itération chaotique va toujours rattraper l'itération standard. Nous appelons cette notion non opérationelle *équité abstraite* par rapport à $\mathcal{F}$.

Définition 11 (équité abstraite) *Une suite $e = (e_i)_I$ vérifie l'équité abstraite par rapport à $\mathcal{F}$ si $\forall i \in I, \exists j \in I,\ e_j \subseteq F_{\mathcal{F}}(e_i)$.*

Théorème 12 *Soit $a = (a_i)_{i\in\mathbb{N}}$ l'itération standard de $F_{\mathcal{F}}$ et $b = (b_i)_{i\in\mathbb{N}}$ une itération chaotique de $\mathcal{F}$ qui vérifie l'équité abstraite, toute deux ayant pour état de départ $d \in \mathcal{SS}$. Alors $lim(b) \subseteq lim(a)$.*

Preuve Il suffit de montrer que $\forall n \in \mathbb{N}$, $lim(b) \subseteq a_n$. Une condition suffisante est $\forall n \in \mathbb{N},\ \exists i \in \mathbb{N}$ tel que $b_i \subseteq a_n$. Cette preuve est faite par induction sur n :

- Pour $n = 0$, $b_0 \subseteq a_0$.
- Soit $n \geq 0$. Il existe $i \in \mathbb{N}$ tel que $b_i \subseteq a_n$. Grâce à la propriété d'équité abstraite, $\exists j \in \mathbb{N},\ b_j \subseteq F_{\mathcal{F}}(b_i)$. Alors, d'après la monotonie de $F_{\mathcal{F}}$, $b_j \subseteq F_{\mathcal{F}}(b_i) \subseteq F_{\mathcal{F}}(a_n) = a_{n+1}$.

La condition appelée équité qui consiste opérationellement à s'assurer que le choix d'un opérateur de réduction n'est jamais ajourné est une condition suffisante pour assurer l'équité abstraite.

Définition 13 (équité) *Une itération chaotique $b = (b_i)_{i\in I}$ est* équitable *si l'une des deux propriétés suivantes est vérifiée :*

- *I est fini et b est complète, ou*
- *$\forall i \in I, \exists j \geq i, \forall f \in \mathcal{F}, \exists k, i < k \leq j$ et $b_k = f(b_{k-1})$.*

Théorème 14 *Si une itération chaotique est équitable alors elle vérifie aussi la propriété d'équité abstraite.*

Corollaire 15 (complétude) *Soient $a = (a_i)_{i\in\mathbb{N}}$ l'itération standard de $F_{\mathcal{F}}$ et $b = (b_i)_{i\in I}$ une itération chaotique équitable, toutes deux partant du même état $d \in \mathcal{SS}$. Si $lim(a)$ est stable alors b est complète.*

Remarque 16 *Dans le cas fini, un résultat plus précis peut être établi :*

Si il existe $n \geq 0$ tel que a_n est stable, alors $\exists m \geq n$ tel que b_m est stable.

Preuve *Il suffit de montrer que $b_m \subseteq a_n$. Grâce à l'hypothèse d'équité, pour tout $i \in [0 \ldots n]$ il existe j_i tel que tous les opérateurs de $\mathcal{F}$ sont appliqués entre les indices i et j_i. Donc, à l'indice j_n, tous les opérateurs de $\mathcal{F}$ sont appliqués au moins n fois. Donc $b_{j_n} \subseteq a_n$.*

On peut remarquer qu'aucune itération chaotique ne peut être complète si l'itération standard n'atteint pas la clôture. Cependant, pour toute itération chaotique équitable, l'approximation obtenue ne dépend pas de l'ordre d'application des opérateurs. Le résultat de confluence des itérations chaotiques est donné par le théorème suivant. Il est obtenu en combinant les théorèmes 9 et 12.

Théorème 17 (itération chaotique, Apt [2]) *Soient $a = (a_i)_{i \in \mathbb{N}}$ l'itération standard de $F_{\mathcal{F}}$ et $b = (b_i)_{i \in I}$ une itération chaotique équitable, toute deux ayant pour état de départ $d \in \mathcal{SS}$. Alors $lim(b) = lim(a)$.*

Ce résultat de confluence montre que deux itérations chaotiques équitables ont la même limite qui est aussi la limite de l'itération standard. Ces résultats peuvent être établis pour un semi-treilli inférieur pour lequel la limite de toute suite décroissante existe [2]. Un algorithme qui implémente les itérations chaotiques est également donné dans [2].

3 γ-itérations chaotiques

L'un des défauts du formalisme des itérations chaotiques est qu'il dépend d'un mécanisme extérieur pour détecter la stabilité. Cette section présente les γ-itérations chaotiques qui, tout en conservant les propriétés des itérations chaotiques, permettent de détecter la stabilité. Elles sont également un moyen de tirer parti de certaines propriétés des opérateurs de réduction pour représenter l'accélération de la convergence. Par exemple, les opérateurs utilisés pour obtenir l'arc-consistance sont généralement idempotents, donc une fois appliqué, il n'est pas nécessaire de les prendre en compte au pas de calcul suivant. C'est pourquoi les algorithmes existants implantent une structure pour l'ordre d'application des opérateurs de réduction : cela peut être une file comme pour AC-3 [19], un ensemble de files comme pour Gnu-Prolog [8] ou une structure plus complexe qui tient compte de différents événements comme dans le système *Choco* [13] dans lequel les propagations peu coûteuses sont faites en premier. Ces structures peuvent être modélisées par un sous-ensemble de $\mathcal{F}$ appelé ensemble de choix. C'est dans ce sous-ensemble noté γ qu'un opérateur sera choisi par une stratégie donnée pour le pas de calcul suivant. Ces sous-ensembles dépendent de la stratégie choisie, de l'état de recherche courant, de la façon dont les opérateurs de réduction sont appliqués ou même de l'historique du calcul. A cause de ces différentes possibilités pour les choix des paramètres, l'ensemble de choix a une définition très générale.

Définition 18 (ensemble de choix) *Soit d un état de l'espace de recherche. Un sous-ensemble γ de $\mathcal{F}$ est appelé un* ensemble de choix *pour d si $\forall f \in \mathcal{F} - \gamma,\ f(d) = d$.*

Soit $d \in \mathcal{SS}$ tel que $\gamma = \emptyset$, alors $\forall f \in \mathcal{F},\ f(d) = d$. Cette propriété est utilisée pour détecter la stabilité d'un état de recherche. De plus, pour chaque état d, il existe un ensemble minimal $\gamma_d = \{f \in \mathcal{F} \mid f(d) \neq d\}$. Mais il se peut aussi qu'un γ calculé contienne des opérateurs qui ne réduisent pas.

Exemple 19 (file de propagation pour AC-3) *L'implémentation de l'algorithme AC-3 nécessite une file de propagation qui contient les opérateurs à appliquer. Au début, la file contient tous les opérateurs. A chaque pas, un opérateur est choisi. Si une composante d_x est réduite, alors tous les opérateurs qui dépendent de la variable x sont ajoutés dans la file à condition qu'ils n'y soient pas déjà. Etant donné que les opérateurs sont idempotents, l'opérateur choisi n'est pas remis dans la file. De plus, tous les opérateurs dépendants de la variable dont le domaine a été réduit sont* *dans la file, donc les opérateurs qui n'y sont pas ne réduisent pas l'état courant. Mais, d'un autre coté, il peut y avoir des opérateurs dans la file dont l'application ne réduit pas les domaines. La file vide signifie que le point fixe a été atteint.*

Une γ-itération chaotique est une sorte d'itération chaotique pour laquelle à chaque pas, l'opérateur de réduction est choisi dans l'ensemble de choix associé à l'état courant :

Définition 20 (γ-itération chaotique) *Une* γ-itération chaotique *de $\mathcal{F}$ commençant à $d \in \mathcal{SS}$ est une suite $c = (c_i, \gamma_i)_{i \in I}$ avec c_i élément de $\mathcal{SS}$ et γ_i ensemble de choix pour l'état courant c_i tel que $c_0 = d$ et $\forall i \in I^+, \exists f \in \gamma_{i-1}, c_i = f(c_{i-1})$.*

A chaque γ-itération chaotique $c = (c_i, \gamma_i)_{i \in I}$ peut être associée l'itération chaotique $(c_i)_{i \in I}$. Donc, en oubliant les γ_i, les γ-itération chaotiques sont un cas particulier d'itérations chaotiques. Réciproquement, les itérations chaotiques sont des γ-itérations chaotiques pour lesquelles l'ensemble de choix γ_i est toujours égal à l'ensemble $\mathcal{F}$. Le cas où il est détecté que plus aucune réduction ne peut être faite dans une γ-itération chaotique est modélisé par le fait qu'il existe un indice $i \in I$, tel que $\gamma_i = \emptyset$. C'est une condition suffisante pour stopper l'itération. Dans ce cas là, la γ-itération chaotique est complète (i.e. l'itération chaotique associée est complète). Cette condition est seulement suffisante car la limite peut être atteinte à un pas $j \leq i$.

De plus, étant donné que les γ-itérations chaotiques sont un cas particulier d'itérations chaotiques, le lemme 9 et le corollaire 10 restent vrais pour les γ-itérations chaotiques et on peut encore utiliser la notion d'équité abstraite. Mais la notion d'équité peut être raffinéé :

Définition 21 (γ-équité) *Une γ-itération chaotique $c = (c_i, \gamma_i)_{i \in I}$ de $\mathcal{F}$ est γ-équitable si $\forall i \in I,\ \exists j \in I,\ j \geq i$ tel que $\forall f \in \gamma_i \cap \gamma_j,\ \exists k,\ i < k \leq j,\ f \in \gamma_{k-1}$ et $c_k = f(c_{k-1})$.*

La γ-équité assure que, entre les indices i et j, un opérateur présent dans l'ensemble de choix à l'étape i doit être appliqué pour pouvoir être dans l'ensemble de choix à l'étape j. Un condition triviale suffisante pour avoir la γ-équité est d'avoir $\forall i \in I,\ \exists j > i$ tel que $\gamma_i \cap \gamma_j = \emptyset$. Tous les opérateurs de γ_i sont alors utilisés et remplacés par de nouveaux. La γ-équité est une condition suffisante pour avoir l'équité abstraite :

Lemme 22 *Soit $c = (c_i, \gamma_i)_{i \in I}$ une γ-itération chaotique de $\mathcal{F}$ commençant à l'état $d \in \mathcal{SS}$. Si c est γ-équitable alors c est abstraitement équitable.*

Preuve On doit montrer que $\forall i \in I,\ \exists j \in I$ tel que $c_j \subseteq F_{\mathcal{F}}(c_i)$.

Soit $i \in I$. Etant donné que c est γ-équitable, il existe un indice j tel que $\forall f \in \gamma_i \cap \gamma_j,\ \exists k,\ i < k \leq j,\ f \in \gamma_{k-1}$ et $c_k = f(c_{k-1})$. Supposons que $c_j \subseteq \bigcap_{f \in \mathcal{F}} f(c_i)$, c'est à dire que $\forall f \in \mathcal{F}, c_j \subseteq f(c_i)$. Soit $f \in \mathcal{F}$:

- si $f \notin \gamma_i$, alors $c_j \subseteq c_i = f(c_i)$.
- si $f \notin \gamma_j$, alors $c_j = f(c_j) \subseteq f(c_i)$.
- si $f \in \gamma_i \cap \gamma_j$, il existe k tel que $i < k \leq j,\ f \in \gamma_{k-1}$ et $c_k = f(c_{k-1})$. Alors $c_j \subseteq c_k$ car $j \geq k, c_k \subseteq f(c_{k-1})$ et $f(c_{k-1}) \subseteq f(c_i)$ car $k - 1 \geq i$.

Ces résultats permettent d'établir le théorème de confluence :

Théorème 23 *Soient $c = (c_i, \gamma_i)_{i \in I}$ une γ-itération chaotique γ-équitable de $\mathcal{F}$ et $a = (a_i)_{i \in I}$ l'itération standard de $\mathcal{F}$, toute deux partant de l'état $d \in \mathcal{SS}$. Alors :*

$$lim(a) = lim(c)$$

Si de plus $lim(a)$ est stable, alors $lim(a) = lim(c) = Cl_{\mathcal{F}}(d)$.

Une modélisation similaire en intention à l'ensemble γ a été présentée dans [3] où une fonction *update* actualise l'ensemble des opérateurs "actifs". Soit $G \subseteq \mathcal{F}$ l'ensemble des opérateurs de réduction actifs et d l'état courant. Après le choix d'un opérateur $g \in \mathcal{G}$, l'algorithme remplace G par $G - \{g\} \cup update(G, g, d)$. Une fonction similaire est présentée dans [12]. Elle possède un paramètre supplémentaire qui est le domaine qui vient juste d'être réduit par l'application d'un opérateur g. Ces fonctions permettent de modéliser AC-3 et d'autres algorithmes semblables plus directement que notre ensemble γ grâce à leur nature opérationnelle. Cependant, l'ensemble γ est plus déclaratif et plus adapté à nos problèmes. Il permet de modéliser n'importe quelle stratégie opérationnelle, même des stratégies complexes qui prennent en compte l'historique du calcul. Dans [2], les *itérations semies-chaotiques* sont présentées. Une itération est semi-chaotique si tous les opérateurs qui n'apparaissent pas un nombre infini de fois à partir d'un rang i dans l'itération ne réduisent aucun domaine pour tout pas $j \geq i$. C'est un cas particulier de γ-itération chaotique car dès qu'un opérateur ne réduit plus aucun domaine, il peut ne pas être dans l'ensemble de choix et donc peut ne pas être appliqué.

4 Arbres de recherche

4.1 Schéma de calcul

Comme cela a été expliqué, trouver les solutions nécessite l'utilisation de deux techniques différentes. La première (décrite dans les parties précédentes) est la réduction de domaine (ou propagation de contraintes). La seconde est la recherche. Le principe est de se focaliser sur des sous-domaines de la clôture calculée par ($\gamma-$)itérations chaotiques. L'efficacité de la programmation par contrainte dépend de la manière dont ces deux techniques sont combinées. Dans la pluspart des cas, les systèmes propagent les contraintes après tout pas de recherche.

Pour modéliser ces calculs qui combinent les deux techniques, nous présentons les *schémas de calcul*. Ils sont construits à partir d'états de calcul qui contiennent un sous-ensemble G de C, l'ensemble R d'opérateurs de réduction associé et un autre sous-ensemble $S \subseteq C$ de contraintes avec son ensemble L d'opérateurs de réduction utilisé pour se focaliser sur une partie de l'espace de recherche.

Définition 24 (Etat de calcul) *Un* état de calcul *est 4-uplet (G, R, S, L) tel que :*

- *G est un sous-ensemble de C.*
- *$S \subseteq C$ est un ensemble de contraintes.*
- *R et L sont les ensembles d'opérateurs de réduction associés respectivement à G et à S.*

G représente l'ensemble des contraintes à résoudre et S est l'ensemble des contraintes générées par la technique de recherche.

Soit D la famille de domaines initiale. A chaque état de calcul (G, R, S, L) est associé la clôture $C_{R \cup L}(D)$ appelée *domaine de l'état de calcul*. Les pas de réduction et de recherche sont modélisés par des transitions entre états de calculs. Il existe deux sortes de transitions.

Définition 25 (Transition) *Soit $c \in C$. Un couple $((G_1, R_1, S_1, L_1), (G_2, R_2, S_2, L_2))$ d'états de calculs est soit une r-transition si $G_2 = G_1 \cup \{c\}$ (et donc $R_2 = R_1 \cup R_c$ avec R_c ensemble d'opérateurs de réduction associé à la contrainte c), soit une s-transition si $S_2 = S_1 \cup \{c\}$ (et donc $L_2 = L_1 \cup R_c$).*

Une r-transition consiste à prendre en compte une nouvelle contrainte de C. Cela correspond à un pas de réduction, c'est à dire à un calcul de clôture. Une s-transition modélise un pas de recherche. Une nouvelle contrainte est ajoutée pour se focaliser sur une partie de l'espace de recherche dans laquelle les solutions vont être cherchées.

Certains états de calcul jouent un rôle particulier.

Définition 26 (états de calcul extrêmes) *Soit* (G, R, S, L) *un état de calcul et* $d = (d_x)_{x \in V}$ *son domaine associé.*

- *Si il existe* $x \in V$ *tel que* $d_x = \emptyset$ *alors l'état de calcul est dit* échec.
- *Si* $\forall x \in V,\ d_x$ *est un singleton alors l'état de calcul est dit* solution.

Un état extrême *est un état échec ou solution.*

Etant donné qu'un état échec a un domaine associé qui a une composante d_x vide, il ne peut y avoir de solutions pour le CSP G dans la partie de l'espace de recherche restreinte par S. Un état solution a un domaine associé qui est un état singletonique stable. C'est donc une solution. Les schémas de calcul permettent de décrire un calcul composé de pas de recherche et de réduction.

Définition 27 (schéma de calcul) *Un schéma de calcul est une suite finie d'états de calcul tel que :*

- *tout couple d'états de calcul consécutifs est une transition.*
- *si il existe un état de calcul extrême alors c'est le dernier.*

Il est important de remarquer qu'un schéma de calcul peut se terminer par un état de calcul qui ne soit pas extrême. Dans ce cas là, si l'état de calcul est (G, R, S, L), nous montrerons que son domaine est bien un sur-ensemble des solutions du CSP G qui sont présentes dans l'espace de recherche restreint par S. Un schéma de calcul est associé à un CSP $\mathcal{C}$ si son premier état est $(\emptyset, \emptyset, \emptyset, \emptyset)$ avec pour domaine associé D et si le dernier état est un état échec ou vérifie $G = \mathcal{C}$.

A chaque état de calcul (G, R, S, L), on peut associer un domaine et une γ-itération chaotique de $R \cup L$ à partir du domaine associé à l'état de calcul précédent. La concaténation de toutes ces γ-itérations chaotiques est un *calcul*. La correction du calcul se formalise par le théorème suivant :

Théorème 28 (Correction) *Soit* $(C_i, R_i, S_i, L_i)_{i \in I}$ *avec* $I = [1..n]$ *un schéma de calcul tel que* D *soit le domaine associé au premier état. Si* s *est un solution de* C_n *telle que* $s \in \Pi C_{L_n}(D)$ *alors* $s \in \Pi C_{R_n \cup L_n}(D)$.

Preuve Evident (les opérateurs de réduction sont corrects).

4.2 Arbre de recherche

Pour s'assurer de la complétude, les schémas de calcul doivent être organisés de telle façon qu'aucune solution ne soit perdue lors d'un pas de recherche ce qui conduit à une organisation arborescente :

Définition 29 (arbre de recherche) *Un arbre de recherche est composé de nœuds qui sont des états de calcul. Il existe deux sortes de nœuds :*

- *un nœud réduction est un nœud* $n = (C, R, S, L)$ *qui a seulement un fils* $n' = (C \cup \{c\}, R \cup R_c, S, L)$ *où* $c \in C$ *est une contrainte,* R_c *est l'ensemble d'opérateurs de réduction associé à* c *et* (n, n') *est une c-transition.*

$$\begin{array}{c}(G, R, S, L)\\ |\\ (G \cup \{c\}, R \cup R_c, S, L)\end{array}$$

- *un nœud partition est un nœud $n = (G, R, S, L)$ tel que :*
  - *chaque transition construite avec un de ses fils est une l-transition*
  - *si $n_1, \dots n_p$ sont les p fils de n, avec $n_i = (G, R, S \cup \{c_i\}, L \cup R_{c_i})$ où $c_i \in C$ est une contrainte et R_{c_i} son ensemble d'opérateurs de réduction associé, alors $\bigcup_{i \in [1..p]} c_i$ est équivalente à la contrainte* vraie.

$$(G, R, S, L)$$

$$(G, R, S \cup \{c_1\}, L \cup R_{c_1}) \quad \dots \; (G, R, S \cup \{c_i\}, L \cup R_{c_i}) \quad \dots \; (G, R, S \cup \{c_p\}, L \cup R_{c_p})$$

Contrairement aux schémas de calculs où une seule partie de $\mathcal{SS}$ est conservée, ici aucune n'est perdue et donc toutes les solutions sont conservées dans les différents domaines des fils. De plus, chaque branche d'un arbre de recherche correspond à un schéma de calcul et vérifie donc les mêmes propriétés : un nœud qui est un état de calcul extrème est une feuille de l'arbre. Il y a donc trois sortes de feuille : solution, échec et celles qui sont des états de calcul non extrèmes. Un arbre de recherche est associé à un CSP $\mathcal{C}$ si toutes ses branches le sont, c'est à dire si $(\emptyset, \emptyset, \emptyset, \emptyset)$ est le nœud racine et D est son domaine associé et si toutes les feuilles non échec (G, R, S, L) vérifient $G = \mathcal{C}$. Maintenant, la complétude des arbres de recherche peut être formalisée par :

Théorème 30 (complétude) *Soit $\mathcal{T}$ un arbre de recherche associé à un CSP $\mathcal{C}$. Pour toute solution s de $\mathcal{C}$, il existe au moins une feuille de $\mathcal{T}$ dont le domaine associé d vérifie $s \in \Pi d$.*

Preuve Soient n un nœud (C, R, S, L) et $\{s_1, \dots, s_q\}$ les solutions de C dans $\Pi C_{R \cup L}(D)$. Si n est un nœud réduction, étant donné que les opérateurs de réduction sont corrects, le domaine associé à son fils contient $\{s_1, \dots, s_q\}$. Si n est un nœud partition, soit $d_1, \dots, d_p$ les domaines associés aux p fils de n. Vu que $\bigcup_{i \in [1..p]} d_p = C_{R \cup L}(D)$, alors $\forall i \in [1..q]$, $\exists j \in [1..p]$ $s_i \in \Pi d_j$.

Dans un cadre de théorie de la preuve, [1] présente des règles qui permettent de construire des arbres de preuves qui correspondent ici aux arbres de recherche.

4.3 Labeling

Comme seulement les domaines finis sont pris en compte, les pas de recherche sont le plus souvent faits en pratique par labeling. Nous considérons ici que cela consiste à partitionner le domaine d'une seule variable. Cette notion générale de labeling nous permet de prendre en compte le labeling classique (qui consiste à instancier la variable avec toutes ses valeurs) mais aussi les techniques de *splitting*. Donc seulement les contraintes de C dont la forme générale est $x \ in \ E$ ($E \subseteq D_x$ fixé) sont utilisées pour procéder à un pas de recherche. Ces contraintes ont seulement un opérateur de réduction associé appelé *opérateur de labeling*.

Définition 31 (opérateur de labeling) *Soient x une variable et $E \subseteq D_x$. L'*opérateur de labeling $f_{x=E}$ *associé à la contrainte* x in E *est la fonction définie par :*

$$\forall d \in \mathcal{SS}, \; f_{x=E}(d)_y = d_y \; si \; y \neq x \; et \; f_{x=E}(d)_x = d_x \cap E.$$

Les opérateurs de labeling sont monotones et contractants. Mais contrairement aux opérateurs de réduction, le but des opérateurs de labeling est de focaliser la recherche sur une partie de l'espace de recherche qui peut ne contenir qu'une partie des solutions, voire aucune. Donc, les opérateurs de labeling sont des opérateurs de réduction “incorrects”. Dans ce contexte, incorrect signifie que quand un opérateur de labeling est appliqué, il n'y a pas moyen de savoir si toutes les solutions (ou même une partie) sont conservées. Comme seulement un opérateur de labeling est associé à chaque contrainte de l'ensemble S, les états de calcul peuvent être simplifiés et représentés par un triplet (C, R, L). Cette simplification permet également de simplifier la définiton des nœuds partitions.

Définition 32 (nœud labeling) *Soit D le domaine associé à la racine d'un arbre de recherche. Un nœud (G, R, L) est un* noeud labeling *si il existe une variable $x \in V$ telle que ses fils ont la forme générale $n_1 = (G, R, L \cup \{f_{\{x=E_1\}}\}), \dots, n_p = (G, R, L \cup \{f_{\{x=E_p\}}\})$ avec $\bigcup_{i \in [1..p]} E_i = C_L(D)$.*

Si d est le domaine associé au nœud labeling, $d = C_{R \cup L}(D) \subseteq C_L(D)$, donc des fils de n pourront être des feuilles échec. Une optimisation naturelle est d'avoir $\bigcup_{i \in [1..p]} E_i = d$. De plus, pour que le labeling ait une influence sur la propagation, la variable x est choisie parmis les éléments de $var(G)$. Nous allons maintenant montrer certaines propriétés des opérateurs de labeling.

Propriété 33 *Soient $f_{x=E}$ un opérateur de labeling et g une composition d'opérateurs de réduction alors :*

$$g \circ f_{x=E} = f_{x=E} \circ g \circ f_{x=E}.$$

Preuve Soit $d \in \mathcal{SS}$. $g \circ f_{x=E}(d) = g(d') = d''$ avec $d'_y = d_y$ pour $y \neq x$ et $d'_x = d_x \cap E$. Mais $d''' = f_{x=E}(d'')$ car $d'' \subseteq d'$ et donc $g \circ f_{x=E}(d) = f_{x=E} \circ g \circ f_{x=E}(d)$.

Grâce à ce résultat, on peut établir le théorème suivant :

Théorème 34 *Soient $d \in \mathcal{SS}$, L un ensemble d'opérateurs de labeling et R un ensemble d'opérateur de réduction. Soit $\mathcal{L}(d) = \bigcap_{l \in L} l(d)$. Alors :*

$$C_{R \cup L}(d) = C_R(C_L(d)) = C_R(\mathcal{L}(d)).$$

Preuve Soient $A = C_{R \cup L}(d)$, $B = C_R(C_L(d))$ et $C = C_R(\mathcal{L}(d))$.
$A \subseteq C_L(d)$ car $\forall f \in L,\ f(A) = A$. De plus, l'opérateur de clôture C_R est monotone donc $C_R(A) \subseteq C_R(C_L(d))$. Mais $C_R(A) = A$ d'où $A \subseteq B$.

D'après la propriété 33, $C_L(d) = \mathcal{L}(d)$ donc $B = C$.

C est stable par R et d'après la propriété 33, C est stable par L donc $C \subseteq A$.

Ce théorème montre que la clôture d'un état de l'espace de recherche par un ensemble d'opérateurs de labeling L et par un ensemble d'opérateurs de réduction R est égale à la clôture par R de la clôture par L de ce même état. Donc, la clôture par L pourrait être calculée en premier et alors L ne serait plus nécessaire. Ceci correspondrait à la stratégie “generate and test” inefficace en pratique. De plus, la clôture par L peut être obtenue par une simple application de de tous les opérateurs de L. Cette caractérisation du résultat du schéma de calcul est l'analogue de la propriété appelée “indépendance par rapport à la règle de choix” dans la résolution SLD.

L'exemple suivant montre que ce théorème n'est plus vérifié quand un pas de recherche est effectué à l'aide d'une contrainte quelconque au lieu d'une contrainte de la forme “x *in* E”.

Exemple 35 *Voici un CSP avec trois variables x, y et z. Le domaine des variables est $[0 \dots 2]$. Les contraintes sont $x < y$ et $y < z$. Pour effectuer un pas de recherche, la contrainte $x = y$ est utilisée. L'arc consistance sert à réduire les domaines. Si un pas de recherche est fait en premier, les domaines ne sont pas réduits. Quand les* *pas de réduction sont faits, le domaine de x est réduit à $\{0\}$, le domaine de y à $\{1\}$ et celui de z à $\{2\}$. D'un autre coté, si les pas de réduction sont faits en premiers, quand le pas de recherche est fait, le domaine des variables devient vide car pour la seule valeur du domaine de x, il n'existe pas de valeur v dans le domaine de y (qui est réduit à $\{1\}$) tel que $0 = v$.*

5 Conclusion

Le but de ce cadre est de donner un modèle pour l'ordonnancement de règles dans un solveur. Ce modèle est basé sur les itérations chaotiques [2]. Grâce à l'introduction de l'ensemble de choix γ, n'importe quelle stratégie d'ordonnancement qui utilise des heuristiques basées par exemple sur des propriétés des contraintes, ... peut être vue comme une instance particulière de notre formalisme. De plus, ce cadre permet de prendre en compte les méthodes de recherche. La recherche se fait à l'aide d'un ensemble de contraintes et de son ensemble d'opérateurs "incorrects" associé. Cet ensemble permet de concentrer la réduction de domaine sur des parties bien définies de l'espace de recherche. Les arbres de recherche permettent de décrire l'interaction entre les méthodes de réduction et les méthodes de recherche au sein d'un même solveur. La méthode de recherche particulière appelée labeling est également traitée en détail. Enfin, l'équité des stratégie d'ordonnacement qui est une condition suffisante pour s'assurer de la convergence est décrite de façon opérationelle. Ce formalisme qui modélise les solveurs pourra être utilisé pour l'analyse de programme avec contraintes et en particulier pour l'obtention de traces de calculs [10], [7].

Références

[1] K. R. Apt. A proof theoretic view of constraint programming. *Fundamenta Informaticae*, 34(3) :295–321, June 1998.

[2] K. R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1-2) :179–210, 1999.

[3] K. R. Apt. The role of commutativity in constraint propagation algorithms. *ACM Transactions on Programming Languages and Systems*, 22(6) :1002–1036, 2000.

[4] Frédéric Benhamou. Heterogeneous constraint solving. In Michael Hanus and Mario Rodríguez-Artalejo, editors, *Algebraic and Logic Programming, 5th International Conference, ALP'96*, volume 1139 of *lncs*, pages 62–76, Aachen, Germany, sep 1996. Springer.

[5] Daniel Chazan and Willard Miranker. Chaotic relaxation. *Linear algebra and its applications*, 2 :199–222, 1969.

[6] Patrick Cousot and Radhia Cousot. Abstract interpretation : A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Principles of Programming Languages*, pages 238–252, 1977.

[7] Pierre Deransart, Francois Fages, Narendra Jussien, Ludovis Langevine, and Raphaël Martin. Débogage dynamique de programmes avec contraintes sur les domaines finis : état et perspectives. In *Délivrable projet OADymPPaC*, page D2.1.1.1, http ://contraintes.inria.fr/OADymPPaC, 2002.

[8] Daniel Diaz and Philippe Codognet. The GNU Prolog system and its implementation. In *ACM Symposium on Applied Computing*, volume 2, pages 728–732, Villa Olmo, Como, Italy, 2000.

[9] François Fages, Julian Fowler, and Thierry Sola. Experiments in reactive constraint logic programming. *Journal of Logic Programming*, 37(1-3) :185–212, 1998.

[10] Gérard Ferrand, Willy Lesaint, and Alexandre Tessier. A model of constraint solvers by chaoticiteration adapted to value withdrawal explanations. In *Délivrable projet OADymPPaC D1.1.1*, http ://contraintes.inria.fr/OADymPPaC, 2001.

[11] Nissim Francez. *Fairness.* Springer-Verlag, 1986.

[12] Laurent Granvilliers and Eric Monfroy. Constraint propagation : between abstract models and ad-hoc strategies. In Rina Dechter, editor, *Principles and Practice of Constraint Programming*, volume 1894 of *Lecture Notes in Computer Science*, pages 505–509, Singapore, September 18-21 2000. Springer. short paper.

[13] François Laburthe and the OCRE project. Choco : implementing a CP kernel. In *TRICS, Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, Technical report TRA9/00, Singapore, sep 2000.

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

[15] Ugo Montanari and Francesca Rossi. Constraint relaxation may be perfect. *Artificial Intelligence*, 48 :143–170, 1991.

[16] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[17] M. H. van Emden. Value constraints in the CLP scheme. *Constraints*, 2 :163–183, 1997.

[18] Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[19] D.L. Waltz. *Understanding line drawings in scenes with shadows*, pages 19–91. McGraw Hill, 1975.