Expressions Itérées en Programmation par Contraintes

Lucas Bordeaux, Youssef Hamadi, Claude-Guy Quimper, Horst Samulowitz

Expressions Itérées en Programmation par Contraintes

Lucas Bordeaux[1] **Youssef Hamadi**[1] **Claude-Guy Quimper**[2] **Horst Samulowitz**[3]

[1]Microsoft Research, Cambridge, UK
[2]École Polytechnique de Montréal, Canada
[3]University of Toronto, Canada
{lucasb,youssefh}@microsoft.com, quimper@alumni.uwaterloo.ca,horst@cs.toronto.edu

Résumé

Pour exprimer certains problèmes avancés d'optimisation et de décision, il est utile d'imposer des contraintes sur des expressions complexes mettant en œuvre des opérations imbriquées comme $\sum_i$, $\min_i$, *etc.* Par exemple, nous avons typiquement besoin de ce type d'expressions pour écrire l'*espérance* d'une fonction dont les paramètres incluent des variables aléatoires. Nous montrons que les expressions itérées représentent une construction claire et naturelle permettant d'exprimer une large palette de problèmes qui sont hors de portée des résolveurs CP classiques, notamment des problèmes de programmation par contraintes stochastique. Nous présentons des résultats préliminaires qui montrent comment ces constructions peuvent s'intégrer dans un résolveur CP.

Abstract

To state advanced optimization and decision problems, it is desirable to impose constraints on complex expressions that involve nested "iterated" operations like $\sum_i$, $\min_i$, *etc.* For instance, we typically need such expressions to write down the expected value of a function whose inputs include random variables. We argue that iterated expressions provide a clear and natural construct for expressing a wide range of problems that are beyond the scope of classical CP solvers, notably problems in stochastic constraint programming. We present preliminary results that demonstrate how these constructs can be integrated within a CP solver.

1 Introduction

Cet article présente un nouveau type d'expressions appelées *expressions itérées* et démontre que leur utilisation dans un cadre de programmation par contrainte autorise la résolution d'un certain nombre de problèmes difficiles à modéliser par des outils classiques. En particulier, les expressions itérées s'avèrent adaptées à la résolution de problèmes stochastiques sous contraintes.

1.0.1 Les expressions itérées

Etant donne une opération binaire $\oplus : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ associative et commutative, nous appelons *application itérée* de $\oplus$ une expression de la forme :

$$\bigoplus_{x \in D} e(x), \tag{1}$$

où $D = \{v_1 \dots v_d\}$ est un ensemble fini de valeurs et $e(x)$ une expression dépendante de (x). Une telle expression représente de façon compacte la formule :

$$e(v_1) \oplus \cdots \oplus e(v_d) \tag{2}$$

Des exemples typiques d'expression itérées sont des sommes de la forme $(\sum_{x \in D} e(x))$, minima $(\min_{x \in D} e(x))$ et maxima[1] $(\max_{x \in D} e(x))$. Si le domaine d'une variable est Booléen, minima et maxima correspondent aux conjonctions et disjonctions. Elles se notent $\bigwedge_{x \in D} e(x)$ et $\bigvee_{x \in D} e(x)$ ou, de façon alternative, en utilisant des quantificateurs. En effet, les operateurs itérés ont plusieurs point communs avec les quantificateurs bornés : ils permettent d'écrire de façon succincte une formule ou une expression qui devrait être déroulée **pour toutes** les valeurs d'un domaine comme présenté dans l'equation 2. Dans l'ab-

[1]Les produits itérés $\prod_{x \in D} e(x)$ peuvent aussi être considérés mais nous nous intéressons plus particulièrement aux $\sum$, min et max dans ce travail.

solu, un tel déroulement aboutirait souvent à des formules exponentiellement larges. Cette observation est à la base de notre travail qui vise à fournir des mécanismes de raisonnement avancés sur les expressions itérées sans avoir à les dérouler entièrement.

Par extension, nous appellerons *contrainte itérée*, une contrainte imposant qu'une expression itérée soit plus petite, plus grande ou égale à une variable. Nous clarifions cette notion par différents exemples.

Exemple 1 *Un example d'expression itérée* fermée *(i.e., sans variable libre) correspond a* $\sum_{x\in[1,3]}\sum_{y\in[1,3]} x + y$. *L'évaluation d'expressions fermées correspond toujours à une constante, 36 dans cet exemple.*

Par opposition, dans la contrainte :

$$\sum_{y_1\in\{0,1\}} \cdots \sum_{y_{10}\in\{0,1\}} x\cdot(y_1+\cdots+y_{10}) \quad \leq 5000 \quad (3)$$

l'expression de gauche possède une variable libre, x. *Sa valeur est donc une fonction de* x, $x \mapsto 5120\cdot x$, *qui a pour intervalle* $[0, 5120]$ *si* $x \in [0, 1]$. *Propager cette contrainte correspond à raffiner l'intervalle de sa variable libre, par exemple en détectant que* $x \leq 0$. *Comme nous le verrons par la suite, la pose de contraintes est toujours possible dans une expression e.g.,* $\sum_x \min_{y:y\geq 2x+1} f(x, y)$.

1.0.2 Objectifs de cet article

Dans cet article, nous nous intéressons à l'évaluation de l'intérêt des expressions itérées (EIs) en programmation par contraintes. Les expressions itérées s'expriment de façon naturelle. Ce simple fait nous laisse penser que les EIs représentent plus qu'une spécialisation du formalisme contraintes. Nous pensons qu'au contraire, pour certaines applications elles représentent sans doute le seul mécanisme viable de modélisation.

Les problèmes qui ont nourri notre réflexion sur les expressions itérées proviennent de domaines d'application mettant en œuvre un certain degré d'incertain et/ou de raisonnement contre l'adversaire. Ces problèmes nécessitent souvent l'utilisation de variables non standard, dont la valeur est fixée par un agent extérieur ou de façon aléatoire (*i.e.,* variables aléatoires).

La section suivante liste de tels problèmes et présente notamment le domaine qui a motivé cette recherche : l'addition d'une capacité de raisonnement dynamique aux moteurs de workflows.

1.0.3 Détail de nos contributions

En section 2 nous donnons différents exemples d'usage des contraintes itérées ; En section 3 nous présentons notre approche de façon formelle et nous étudions sa capacité d'expression ; En section 4 nous considérons les problèmes de l'évaluation et de la propagation des contraintes itérées ; En section 5 nous montrons comment une méthode de recherche classique peut être étendue pour résoudre les contraintes itérées. Des remarques générales et une conclusion sont finalement présentées en section 6.

2 Utilisation des Expressions Itérées

2.1 Variables Aléatoires

Une application importante des sommes itérées est lie à l'utilisation de variables *aléatoires*. Supposons une telle variable Y dont le domaine correspond a un intervalle discret D_Y. Nous connaissons la probabilité $\Pr(Y = a)$ pour chaque valeur du domaine. Si une expression f dépend de Y, la valeur attendue de f représente une certaine quantité que l'on doit pouvoir utiliser dans la pose de contraintes ou dans la définition de fonction objectives. Cela s'exprime :

$$\sum_{a\in D_Y} \Pr(Y = a)\cdot f(a) \quad (4)$$

Quand le domaine D_Y est relativement petit, *e.g.,* $\{0, 1\}$, nous pouvons facilement et légitimement dérouler l'expression en $\Pr(Y = 0)\cdot f(0) + \Pr(Y = 1)\cdot f(1)$.

Si au contraire, le domaine est grand, la taille de l'expression peut croitre de manière considérable. Mais le point le plus important est que le déroulement devient réellement exponentiel quand plusieurs variables aléatoires sont combinées. Par exemple, la valeur attendue de la fonction $f(Y_1 \ldots Y_n)$ de n variables Booléennes est :

$$\sum_{a_1\in\{0,1\}} \cdots \sum_{a_n\in\{0,1\}} \left(f(a_1 \ldots a_n)\cdot \prod_{i\in 1..n} \Pr(Y_i = a_i) \right) \quad (5)$$

Le déroulement d'une telle somme mettrait en œuvre 2^n termes. Ce simple exemple démontre que les expressions itérées ne peuvent pas toujours être pratiquement déroulées. Nous devons mettre en œuvre des mécanismes capables de raisonner directement sur l'expression.

Différentes mesures statistiques peuvent aussi être exprimées avec les EIs. Par exemple, des contraintes légèrement plus complexes pourraient aisément définir la *variance* de $f(Y_1 \ldots Y_n)$. Dans la suite, nous considérons le cas plus simple de l'Esperance mathématique.

2.2 Un exemple détaille : modéliser les workflows

A présent, voyons plus en détail comment les variables aléatoires apparaissent dans une classe particulière d' applications : les moteurs de workflows. Les moteurs workflows sont déjà employés couramment pour commander l'exécution des processus dans les entreprises, et ils sont sur le point de devenir omniprésents [2].

Dans l'exécution de beaucoup de workflows, notamment ceux impliquant des ressources humaines, des décisions intelligentes doivent être prises (*e.g.,* allocation d'une tache à une ressource). D'ordinaire, ces décisions sont implémentées de façon naïve par le système, ou faites par les utilisateurs qui interagissent avec le système. Par exemple une règle simple peut décider quel employé a affaire avec un client particulier. L'intégration d'une aide a la décision basés sur la programmation par contraintes a le potentiel de rendre ce procédé de décision plus facile et plus efficace, et, dans un projet parallèle, nous travaillent à de tels systèmes.

Les fonctions que nous voulons optimiser dans le contexte des workflows nous donnent de bons exemples de l'utilisation des expressions itérées. Nous manquons de place pour présenter la notion de workflow de manière détaillée. A la place, nous motiverons simplement notre utilisation des expressions itérées en esquissant un exemple simple lié à l'allocation de ressource. Un prochain article décrira notre projet de moteur de workflow à base d'EIs dans plus de profondeur.

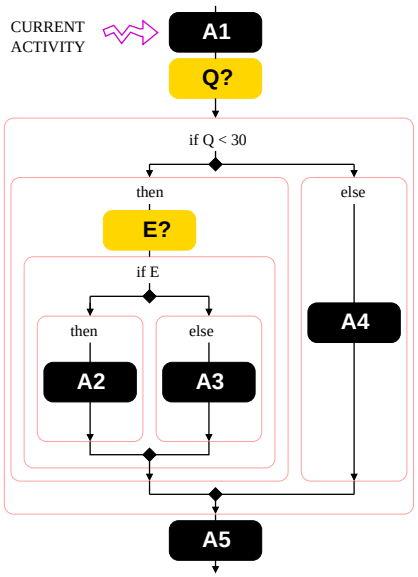

FIG. 1 – A workflow

[2]Un signe de la généralisation de l'usage de ces moteurs est leur intégration native au nouveau système d'exploitation Microsoft Windows Vista.

2.2.1 Un exemple simple

La figure 1 représente un workflow impliquant quelques activités de base $A_1 \dots A_5$. Ce sont des activités qui devront être exécutées par l'entreprise, selon la branche qui est choisie. Nous avons un ensemble de ressources R et l'exécution d'une des activités mobilisera une des ressources. La décision allouant une activité a une des ressource est dénotée X_i ; le coût d'assigner cette ressource à l'activité A_i est $f_i(X_i)$. Maintenant, dans la Fig. 1 nous avons également 2 activités (couleur claire) qui comprennent un point d'interrogation. Ces activités représentent les décisions qui ne sont pas prises par la compagnie mais par le client, et l'effet de ces décisions sera d'assigner une variable dont nous ne connaissons a priori que la distribution de probabilité sur l'ensemble des valeurs possibles. Par exemple, la variable Q représente la quantité de marchandises que le client commandera (s'étendant au-dessus de $[1, 100]$, avec une distribution indiquée), tandis que la variable E représente un Booléen et sera vraie ssi le client choisit une livraison express (qui a par exemple la probabilité 0.4).

2.2.2 Coût attendu de l'exécution d'un workflow

Maintenant, supposons que A_1 est l'activité courante. Nous décidons de l'affectation d'une ressource a A_1 en considérant les conséquences de notre décision non seulement à court terme (*e.g.,* coût d'allouer une ressource particulière en ce moment), mais également pour le reste de l'exécution (*e.g.,* allouer une ressource particulière à A_1 peut affecter sa disponibilité dans d'autres activités). Une quantité importante que nous voudrions manipuler est le coût (minimal) possible lors de l'exécution de l'ensemble des activités restantes du workflow. Ce coût est obtenu en composant les coûts de base $f_1 \dots f_5$, mais ceux-ci doivent être agrégés et pesés d'une manière qui reflète la structure du déroulement des opérations. Le coût prévu peut par exemple être écrit comme :

$$\min_{r_1 \in R} \left(f_1(r_1) + \sum_{a \in D_Q} \Pr(Q = a) \begin{pmatrix} \textit{if } a < 30 \textit{ then} \\ \boxed{\text{M}} \\ \textit{else} \\ \min_{r_4} f_4(r_4) \end{pmatrix} + \min_{r_5 \in R} f_5(r_5) \right) \tag{6}$$

ou l'expression M, représente le cout attendu du sous-workflow "E ? ; if E then A_2 else A_3", est :

$$M \equiv \sum_{b \in \{0,1\}} \Pr(E = b) \cdot \left(\textit{if } b \textit{ then} \min_{r_2 \in R} f_2(r_2) \textit{ else } \min_{r_3 \in R} f_3(r_3) \right) \tag{7}$$

Comme le domaine de E est petit, nous pourrions dérouler M en :

$$\Pr(E=1)\cdot\min_{r_2\in R} f_2(r_2)+\Pr(E=0)\cdot\min_{r_3\in R} f_3(r_3) \quad (8)$$

mais, comme présente plus haut, cette approche est complètement irréaliste dans le cas général.

Remarquons que le *if-then-else* employé devrait être lu en tant que fonctionnel " if " semblable au *cond ? value1 : value2* du langage de programmation C *e.g.,* définition d'une valeur par cas. Il est clair que la syntaxe des expressions itérées utilisées ici est intimement liée à la structure du workflow. En effet, en dérivant l'expression correspondant au workflows nous avons appliqué de manière systématique un ensemble de règles : le coût d'une expression *if* est défini par cas (des règles semblables peuvent être définies pour d'autres opérateurs de base *e.g.,* exécution parallèle, etc.) ; chaque fois que nous devons assigner une ressource, nous calculons systématiquement un *min* des coûts possibles, alors que chaque fois que nous rencontrons une décision externe nous calculons une somme pondérée sur les coûts possibles.

2.2.3 Considérer les qualifications et les disponibilités des ressources

Dans l'expression que nous avons obtenue rien nous ne nous empêche d'assigner n'importe quelle ressource à n'importe quelle activité. Dans la pratique, nous devons typiquement tenir compte des disponibilités des ressources et de leurs qualifications, qui peuvent être compatibles ou incompatibles avec les conditions requises par les activités. En d'autres termes nous avons des contraintes sur les indices itérés que nous dénoterons $\bigoplus_{x:C} e(x)$ (itération sur tous les x *satisfaisant* C). The cout attendu étant alors :

$$\min_{r_1:C_1(r_1)}\left(f_1(r_1)+\sum_{a\in D_Q}\Pr(Q=a)\left(\cdots\right)+\min_{r_5:C_5(r_1,r_5)} f_5(r_5)\right) \quad (9)$$

Nous appelons une telle EI une expression itérée *contrainte*. L'idée est similaire au mécanisme de quantificateurs restreints de [2].

2.2.4 Utiliser les dépendances entre probabilités

Dans beaucoup de cas les probabilités ne sont pas indépendantes. Par exemple il est très probable que le choix d'un mode de livraison "express" dépende de la quantité qui est commandée. Pour exprimer des probabilités conditionnelles nous n'avons qu'à remplacer un terme simple par une expression légèrement plus complexe.

Par exemple, une façon simple d'exprimer $\Pr(E=b)$ serait l'expression $0.4b+0.6(1-b)$, qui s'évalue a 0.4 pour la valeur 1 et 0.6 pour la valeur 0.

Pour une probabilité conditionnelle $\Pr(E=b|Q=a)$ nous avons besoin d'une expression qui encode essentiellement une table de probabilités , *e.g.,* "*if* $Q>5$ *then* $0.2b+0.8(1-b)$ *else* 0.5".

2.3 Autres applications

Dans tous les exemples précédents nous avons suggéré comment employer l'opérateur $\sum$ chaque fois qu'une variable reflète un événement décidé de façon aléatoire ou non contrôlée (variables aléatoires). Un autre cas non-classique et intéressant correspond aux situations ou les variables représentent la décision des adversaires ou des concurrents (*e.g.,* [5]). Ces cas seront typiquement représentés par un max (si notre propre but est minimiser la fonction). Dans ce cas nous employons généralement une approche pessimiste supposant que l'adversaire tendra à réduire au minimum notre propre avantage, ce qui peut être exprimé en utilisant la dualité entre min et max : par exemple le Max sera employé pour les variables de l'adversaire si nos variables emploient le Min. Nous obtenons typiquement un objectif de la forme :

$$\min_{a_1\in D_1}\max_{b_1\in D'_1}\cdots\min_{a_n\in D_n}\max_{b_n\in D'_n} f(a_1,b_1,\ldots a_n,b_n) \quad (10)$$

si nous voulons minimiser $f(X_1,Y_1,\ldots X_n,Y_n)$, où les X_is modélisent nos décisions et les Y_is celles de l'adversaire.

Finalement, signalons que les EIs peuvent être utilisées pour modéliser certains cas de problèmes de comptage [6] : si $f(X_1\ldots X_n)$ est une fonction qui retourne un Booléen, alors l'expression itérée $\sum_{a_1\in D_1}\cdots\sum_{a_n\in D_n} f(a_1\ldots a_n)$ représente le nombre de tuples qui s'évaluent à 1.

3 Expressions itérées

Une expression itérée a la forme :

$$\bigoplus_{x_1\in D_1}^{1}\cdots\bigoplus_{x_n\in D_n}^{n} f(x_1\ldots x_n) \quad (11)$$

où chaque $\bigoplus^i$ appartient à $\{\sum,\max,\min\}$ et f une fonction que nous supposons polynôme. Cependant, comme nous l'avons vu dans l'exemple relatif aux workflows, il est préférable ne pas imposer que tous les symboles $\bigoplus$ soit à gauche de la formule. Au contraire, il est important d'autoriser l'emboitement arbitraire de formules construites a partir des symboles $\sum$, max, min, + et $\cdot$.

Ceci réclame des définitions de style inductif au lieu de la forme rigide $\bigoplus_{x_1}^{1} \cdots \bigoplus_{x_n}^{n} f$.

3.1 Définitions

Définition 1 *Les expressions itérées se définissent comme suit :*

1. *Une constante k ou une variable x est une EI ;*
2. *Etant donnée deux EIs e, e', les expressions $e+e'$ et $e \cdot e'$ sont des EIs ;*
3. *Etant donnée une EI e, une variable x libre dans e, et une ensemble de contraintes C relatives a x et l'ensemble des variables libres de e, les expressions $\max_{x:C} e$, $\min_{x:C} e$, et $\sum_{x:C} e$ sont des EIs.*

On suppose que chaque variable x est associée a un domaine fini D_x. Les *variables libres* d'une EI sont définies de manière classique, *i.e.,* comme des variables apparaissant de l'expression non liées a un operateur itéré $\bigoplus_x$.

En l'absence de contrainte C, on peut simplifier la notation $\bigoplus_{x:\varnothing} e$ en $\bigoplus_x e$. L'expression est alors qualifiée de *basique* ; les autres expressions étant qualifiées de *contraintes*. Les types de contraintes autorisés dans C sont ceux fournis par le solveur sous-jacent. Les contraintes typiquement utiles correspondent aux contraintes arithmétiques classiques.

Remarquons que notre definition fait que C peut intégrer l'ensemble des variables sous sa portée. Ceci autorise l'écriture d'expressions telles que $\sum_x \min_{y:y\geq x} f(x,y)$, dans laquelle l'inégalité lie y a une variable précédent dans l'expression, x.

Dans le cas d'une expression contrainte $\bigoplus_{x:C} e$, nous prenons seulement le min, max ou la somme[3] des valeurs du domaine qui satisfont la contrainte.

Une question qui se pose est de savoir quelle valeur donner a l'expression quand aucun x ne satisfait C. Conformément à la semantique normale des domaines vides lors de l'utilisation de solveurs, nous considérons l'expression invalide et retournons un échec. Cette approche a l'avantage d'être bien fondée mathématiquement : un min sur un ensemble vide doit être défini $+\infty$, et cette valeur est n'est pas incluse dans un domaine fini.

[3]Remarquons ici que même si les sommes itérées contraintes sont parfaitement définies, leur utilisation avec des variables aléatoires semble avoir peu d'intérêt : dans une expression de la forme $\sum_{a:C} \Pr(X=a)\cdot e$, à moins que C ne soit vraie pour chaque valeur $a \in D_X$, les probabilités qui sont considérées ne s'additionnent pas pour donner la valeur unité. Pour l'instant, il est difficile de trouver dans quel contexte de telles expressions pourraient se révéler utiles. Notre expz'erience est que nous n'imposons en général de contraintes que sur les min et max.

3.2 Expressivité

3.2.1 Le construct if-then-else.

Dans nos exemples, nous avons utilisé la notation classique *if-then-else*. Il est possible d'étendre la definition Def. 1 pour autoriser nativement les EIs de la forme "*if b then e_1 else e_2*", ou e_1, e_2 sont des EIs et b est une (in)égalité entre EIs.

Nous préférons une autre approche qui nous permet de conserver une definition minimale et de considérer la notation *if* comme du sucre syntaxique.

Le but est, comme nous le verrons en section IV, de définir des règles d'évaluation d'intervalles qui se prêtent naturellement aux polynômes, et qui rendent l'utilisation du construct if moins désirable dans un tel cadre.

Pour encoder les conditionnelles, nous utilisons l'idée que "*if b then l else r*" s'évalue a $b\cdot l + (1-b)\cdot r$. Quand l'expression b n'est pas directement exprimée en utilisant un polynôme, nous pouvons aisément introduire une nouvelle variable et la contraindre pour représenter la condition. Par exemple,

$\sum_{a\in\{0,1,2\}}$ *if $a=2$ then e_1 else e_2* représente du sucre syntaxique pour l'expression :

$$\sum_{a\in\{0,1,2\}} \left(\min_{b:(b\leftrightarrow(a=2))} (b\cdot e_1 + (1-b)e_2) \right) \qquad (12)$$

3.2.2 Exprimer des probabilités

La notation $\Pr(X = a)$ représente une expression donnant la probabilité de chaque valeur a. Par exemple, si X a un intervalle $\{0,1,2\}$, avec des probabilités respectives 0.5, 0.4, and 0,1, la notation $\Pr(X = a)$ correspond a l'expression :

$$\textit{if } a=0 \textit{ then } 0.5 \textit{ else (if } a=1 \textit{ then } 0.4 \textit{ else } 0.1) \qquad (13)$$

3.2.3 Complexité de l'évaluation

L'expressivité vient à un prix, et l'évaluation des expressions itérées est complexe. Dans la table suivante nous listons les résultats[4] qui peuvent être obtenus pour les EIs basiques de la forme $\bigoplus_{x_1} \cdots \bigoplus_{x_n} f$, fonction de f et des types de connecteurs itères autorises. (pour les EIs contraintes, nous sommes invariablement PSPACE-complete). Le suffixe "-c" est un raccourci pour "-complete". Remarquons le caractère traitable des sommes de polynômes à paramètre fixe.

[4]Nous nous concentrons sur le problème de décision : "est-ce qu'une expression s'évalue a v ?". Nous omettons les preuves par manque de place.

opérations iter. Autorisées	restrictions sur fonction f	complexité de l'évaluation
$\sum, \max, \min$	linéaire	$\mathcal{O}(n^2)$
$\sum$	poly. de degré fixe k	$\mathcal{O}(n^{k+1})$
$\min$	poly. de degré fixe k	NP-difficile
$\max$	poly. de degré fixe k	NP-difficile
$\sum$	poly., degré quelconque	♯P-c
$\sum, \min$	poly. quadratique	PSPACE-c
$\sum, \max$	poly. quadratique	PSPACE-c
$\max, \min$	poly. quadratique	PSPACE-c

4 Évaluation Intervalle et propagation

Intégrer les expressions itérées dans un solveur requiert en premier lieu de les intégrer à son mécanisme de propagation. Notre hypothèse est celle d'un mécanisme de *propagation d'intervalles*, tel qu'utilisé par la majorité des résolveurs pour gérer les expressions numériques ; nous considérons donc désormais que les domaines de toutes les variables sont des intervalles. Dans ce contexte, notez qu'une expression itérée s'évalue en un intervalle (*e.g.,* $\sum_{x\in[1,3]} x + y$ est evalué en $[6, 9]$ si $y \in [0, 1]$), alors qu'une expression itérée close s'évalue toujours en une constante (*e.g.,* $\sum_{x,y\in[1,3]} x + y$ s'évalue en 24). En regard aux résultats de complexité mentionnes précédemment, il n'est pas possible de calculer efficacement l'intervalle de valeurs exact d'une expression mais, suivant l'approche classique en arithmétique d'intervalles, nous étudions des moyens efficaces de calculer un intervalle qui approxime l'intervalle de valeurs d'une expression itérée (*i.e.,* l'intervalle calculé sera toujours garanti d'inclure ces valeurs, mais peut être plus large).

4.1 Évaluation Intervalle : Cas Général

L'évaluation intervalle naturelle d'une expression itérée e est notée $\mathtt{I}(e)$ et définie inductivement comme suit :

$$\begin{aligned} \mathtt{I}(\textstyle\sum_{x\in[l,r]} e) &= (r-l+1)\cdot \mathtt{I}(e) \\ \mathtt{I}(\max_{x\in[l,r]} e) = \mathtt{I}(\min_{x\in[l,r]} e) &= \mathtt{I}(e) \end{aligned} \tag{14}$$

L'évaluation des produits et sommes *i.e.,* $\mathtt{I}(e+e')$ et $\mathtt{I}(e\cdot e')$, est définie de la manière habituelle. Le principal avantage de cette méthode est qu'elle peut être utilisée pour des EIs arbitraires (et peut être étendue facilement pour gérer aussi le *if-then-else*). Son défaut est qu'elle donne une évaluation imprécise des sommes : comme on s'interdit de dérouler cette opération, la seule façon d'évaluer x est de considérer son intervalle $[l, r]$, d'évaluer e avec x remplacé par cet intervalle, et d'utiliser le fait que *chaque itération* prendra valeur dans cet intervalle. Nous multiplions donc $\mathtt{I}(e)$ par le nombre de valeurs dans le domaine de x. La conséquence est un effet indésirable de *multiplication de l'imprécision* : la sur-approximation de l'évaluation intervalle est multipliée par la largeur de l'intervalle, comme démontré sur un exemple.

Exemple 2 *Nous considérons une fonction $Y_1 + Y_2 + X$, où Y_1, Y_2 et X sont affectées dans cet ordre et ont pour domaine $D = [1, 3]$; les variables Y_1 et Y_2 sont des variables aléatoires ayant pour distribution de probabilités* 0.6, 0.3 *et* 0.1 *pour les valeurs 1, 2 et 3 ; et X est une variable de décision. L'espérance de la valeur est :*

$$\sum_{a\in D} \Pr(Y_1 = a)\cdot\left(\sum_{b\in D} \Pr(Y_2 = b)\cdot\left(\min_{c\in D}(a+b+c)\right)\right) \tag{15}$$

Par le mécanisme général d'évaluation intervalle, nous obtenons :

$$\begin{aligned} &3\cdot[0.1, 0.6]\cdot 3\cdot[0.1, 0.6]\cdot([1, 3]+[1, 3]+[1, 3]) \\ &= 9\cdot[0.01, 0.36]\cdot[3, 9] \;= [0.27, 29.16] \end{aligned} \tag{16}$$

ce qui, bien entendu, est correct, mais également imprécis : l'intervalle de valeurs réellement possibles est en fait $[4, 4]$.

4.2 Évaluation Intervalle : Cas Particuliers

Nous proposons un algorithme amélioré d'évaluation intervalle pour le cas particulier dans lequel les IEs sont de forme $\bigoplus^1_{x_1}\cdots\bigoplus^n_{x_n} P$. Cette forme apparaît abondamment dans certaines applications, en particulier lors de la modélisation de longues séquences de décisions dans les workflows. Nous supposons que P est un polynôme en forme développée et que les intervalles de valeurs pour chaque x_i sont petits ; par ailleurs chaque $\bigoplus^i$ peut être de forme $\min_v e$, $\max_v e$ ou $\sum_v e$, mais nous autorisons aussi la forme plus générale $\sum_v \Pr(X = v)\cdot e$, qui est fréquente.

4.2.1 Calculs sur les Polynômes.

Etant donnée une EI e de la forme mentionnée, l'idée est qu'au lieu de calculer directement $\mathtt{I}(e)$, nous calculons un polynôme $\mathtt{P}(e)$ représentant l'expression, puis nous appliquons l'évaluation intervalle sur $\mathtt{P}(e)$. Ceci donne en général une évaluation plus précise. Le polynôme est représente en forme développée.

Pour calculer le polynôme $\mathtt{P}(e)$ nous procédons comme suit : Partant de l'expression $e \equiv \bigoplus^1_{x_1}\cdots\bigoplus^{n-1}_{x_{n-1}}(\bigoplus^n_{x_n} P)$, nous éliminons le connecteur itéré situé à l'extrême droite en calculant le polynôme $P' = \bigoplus^n_{x_n} P$. Nous obtenons une expression équivalente $e' \equiv \bigoplus^1_{x_1}\cdots\bigoplus^{n-1}_{x_{n-1}} P'$ qui est de la même forme

que l'expression initiale, et nous répétons le procédé. Lorsque tous les connecteurs sont éliminés nous obtenons un polynôme dont les seules variables sont les variables libres de e.

Pour le calcul du polynôme représentant $(\bigoplus_{x_n}^{n} P)$ a chaque étape, nous procédons comme suit. Si $\oplus^n$ est une somme, nous utilisons simplement l'addition sur les polynômes (nous prenons chaque terme de la forme développée et additionnons leurs coefficients) ; si l'operateur est un min or max, nous remplaçons x_n par son intervalle, obtenant ainsi un polynôme a coefficients intervalles. En résumé $\mathtt{P}(e)$ est donc défini formellement comme suit :

$$\mathtt{P}(\min_{v\in[l,r]} e) = \mathtt{P}(\max_{v\in[l,r]} e) = \mathtt{P}(e)|_{v:=[l,r]}$$

$$\mathtt{P}(\textstyle\sum_{v\in[l,r]} e) = \mathtt{P}(e)|_{v:=l} + \cdots + \mathtt{P}(e)|_{v:=r}$$

$$\mathtt{P}(\textstyle\sum_{v\in[l,r]} \Pr(X=v)\cdot e) = \Pr(X=l)\cdot \mathtt{P}(e)|_{v:=l} + \cdots + \Pr(X=r)\cdot \mathtt{P}(e)|_{v:=r} \quad (17)$$

Ou les additions sont effectuées sur les polynômes, et $e|_{x:=f}$ représente l'expression e dans laquelle chaque occurrence de la variable x est remplacée par l'expression f.

Si w représente la largeur du plus grand intervalle $[l, r]$, la complexité en temps du calcul de $\mathtt{P}(e)$ est polynomiale en n et w, *i.e.,* elle est *pseudo-polynomiale* (polynomial en la largeur, par opposition au temps requis pour développer l'expression, qui est *exponentiel* en n). C'est pourquoi la méthode est restreinte aux domaines de taille raisonnable – quand les domaines sont trop larges la seule solution est d'utiliser le mécanisme général d'évaluation intervalle qui, bien que moins précis, est moins sensible a la taille des domaines. Pour comprendre pourquoi l'intervalle calculé est plus précis, évaluons $\sum_{x\in[1,3]} x^2-2xy+1$: nous obtenons le polynôme $(1+4+9)-2(1+2+3)y+(1+1+1) = 17-12y$. Nous évaluons ensuite ce polynôme par l'évaluation intervalle classique ; par exemple si le domaine de y est $[0, 2]$ nous obtenons $[-7, 17]$. Avec le mécanisme général d'évaluation intervalle nous aurions obtenu $3\cdot([1, 9] - [0, 12] + 1) = [-30, 30]$.

4.2.2 Le Cas Linéaire

Un cas particulièrement intŕesant est lorsque le polynôme P se trouve être linéaire. Les calculs peuvent alors êtres à la fois plus simples et plus précis :

- Pour évaluer $\mathtt{P}(\min_{a\in[l,r]} e)$ dans le cas où $e' = \mathtt{P}(e)$ est linéaire, nous pouvons simplement retourner $e'|_{a:=l}$ si le coefficient de a dans e' est positif, et $e'|_{a:=r}$ dans le cas contraire (la règle pour max est, bien entendu, duale) ;
- Pour evaluer $\mathtt{P}(\sum_{a\in[l,r]} \Pr(X = a)\cdot e)$ lorsque $e' = \mathtt{P}(e)$ est linéaire et égal à $k_0 + k_1x_1 + \cdots + k_nx_n + k_{n+1}a$, nous pouvons simplement calculer $k'_0 + k_1x_1 + \cdots + k_nx_n$, où $k'_0 = k_0 + k_{n+1}\cdot\sum_{a\in[l,r]} \Pr(X = a)\cdot a$.

Pour clore cette section sur l'évaluation spécialisée nous revenons sur notre exemple :

Exemple 3 *Utilisons les règles spécialisées d'évaluation intervalle sur l'exemple 2 :*

$$\begin{aligned}&\textstyle\sum_a \Pr(Y_1 = a)\cdot(\sum_b \Pr(Y_2 = b)\cdot(\min_c(a+b+c)))\\ &= \textstyle\sum_a \Pr(Y_1 = a)\cdot(\sum_b \Pr(Y_2 = b)\cdot(a+b+1))\\ &= \textstyle\sum_a \Pr(Y_1 = a)\cdot(a+0.6\cdot 1+0.3\cdot 2+0.1\cdot 3+1)\\ &= \textstyle\sum_a \Pr(Y_1 = a)\cdot(a+2.5)\\ &= 0.6\cdot 1+0.3\cdot 2+0.1\cdot +2.5 = 4\end{aligned} \quad (18)$$

Les contraintes étant linéaires et aucune variable n'étant libre, nous obtenons un intervalle reduit à une valeur unique.

4.3 Propagation

Ayant défini l'évaluation intervalle, nous pouvons directement utiliser la technique de *Box-Consistance* [3] pour propager une contrainte itérée, *i.e.,* réduire les bornes de ses variables libres comme montré dans l'exemple 1. Brièvement, l'idée à la base de la Box-Consistance est que lorsque nous avons, par exemple, une égalité entre deux expressions $f = g$, nous pouvons tester si une valeur a est consistante pour x en instanciant x à a : si les intervalles ne sont pas compatibles, *i.e.,* si $\mathtt{I}(f|_{x:=a}) \cap \mathtt{I}(g|_{x:=a}) = \varnothing$, alors la valeur a peut être éliminée. Par exemple la contrainte itérée considérée dans l'exemple 1 était $f(x) \le 5000$, avec :

$$f(x) = \sum_{y_1\in\{0,1\}} \cdots \sum_{y_{10}\in\{0,1\}} x\cdot(y_1 + \cdots + y_{10}) \quad (19)$$

La raison pour laquelle $x = 1$ n'est pas Box-Consistant est que $f(1) = 5120$ est d'intersection vide avec $]-\infty, 5000]$.

Nous ne donnerons pas de détail sur la propagation des contraintes présentes dans les (indices des) EIs. La raison en est simple : ces contraintes sont propagées *de la manière classique.* Par exemple si nous avons une expression $\sum_{x\in[1,3]} \min_{y\in[1,3]:y\ge x} f(x, y)$ et si, durant la phase de recherche, x est fixé a 2, ceci se propage sur l'intervalle de valeurs pour y, qui devient $[2, 3]$.

Exemple 4 *Considérons la contrainte :*

$$\sum_{a\in D} \Pr(Y_1 = a)\cdot\left(\sum_{b\in D} \Pr(Y_2 = b)\cdot\left(\min_{c\in D}(x+a+b+c)\right)\right) \ge 10 \quad (20)$$

où Y_1 and Y_2 sont des variables aléatoires ayant pour distribution de probabilitités 0.6, 0.3 *and* 0.1 *pour les valeurs (resp.) 1, 2 et 3, et x a pour domaine* $[0, 10]$. *Par propagation nous déduisons $x \in [6, 10]$. Pour le vérifier, notez qu'instancier x à 1 donne la valeur 5, à 2 donne la valeur 6, etc., et la Box-Consistance trouve une borne inférieure $x = 6$.*

5 Branch & Bound

Nous nous intéressons maintenant au problème de déterminer *exactement* l'intervalle de valeurs attaché à une expression itérée. Les mécanismes étudiés dans cette section sont donc intégrés dans la partie *recherche arborescente* (ou *Branch & Bound*) du résolveur.

5.1 Expressions Itérées de Base

Nous commençons avec les EIs basiques, et supposons pour le moment que nous sommes a un stade de la recherche ou toutes les variables libres ont été instanciées, si bien que l'EI possède une valeur réelle bien définie. Nous détaillons les différents cas possibles d'EI et définissons pour chaque cas une méthode d'évaluation.

Pour une EI de la forme $e_1 + e_1$, nous pourrions simplement évaluer les deux sous-expressions e_1 et e_2 et retourner leur somme. Mais ce mécanisme n'autorise aucune forme d'élagage. Afin d'exploiter l'information obtenue par évaluation intervalle, nous définissons la fonction $eval(e, [inf, sup])$ comme étant dépendante d'un argument supplémentaire—l'intervalle dans lequel la recherche a lieu. La fonction retourne un couple $\langle b, v\rangle$ où b est vrai ssi la valeur de l'expression est incluse dans $[inf, sup]$ et v est la valeur obtenue (v est arbitraire si $\neg b$). Cette approche autorise un réel *Branch & Bound*.

function $eval(e_1 + e_2, [inf, sup])$

$[inf_1, sup_1] \leftarrow \mathtt{I}(e_1) \cap ([inf, sup] - \mathtt{I}(e_2))$
if $[inf_1, sup_1] = \varnothing$ **then return** $\langle false, \_\rangle$
else
  $\langle ok_1, v_1\rangle \leftarrow eval(e_1, [inf_1, sup_1])$
  if $\neg ok_1$ **then return** $\langle false, \_\rangle$
  else
    $\langle ok_2, v_2\rangle \leftarrow eval(e_2, \mathtt{I}(e_2) \cap ([inf, sup] - v_1))$
    return $\langle ok_2, v_1 + v_2\rangle$

L'idée est facile a comprendre sur un bref exemple : si $\mathtt{I}(e_1) = \mathtt{I}(e_2) = [1, 10]$ et si, à ce stade de la recherche arborescente, nous cherchons une somme dans $[20, 25]$, alors les seules valeurs présentant un intérêt pour (par exemple) la branche gauche sont celles incluses dans l'intervalle $[20, 25] - [1, 10] = [10, 24]$ qui, intersecté avec $[1, 10]$, donne $[10, 10]$. Un algorithme similaire peut être défini pour $eval(e_1 \cdot e_2, [inf, sup])$.

Pour min (et, symétriquement, max), nous parcourons l'ensemble des valeurs possibles et, à chaque fois que nous rencontrons une solution, nous réduisons l'une des bornes en conséquence ; par exemple si nous cherchons une valeur minimale dans $[0, 9]$ et trouvons qu'une des branches s'évalue en 4.7, nous pouvons pour le reste de la recherche nous restreindre aux solutions contraintes a l'intervalle réduit $[0, 4.7]$: [5]

function $eval(\min_{x\in[l,r]} e, [inf, sup])$

$\langle solution\_found, val\rangle \leftarrow \langle false, \_\rangle$
foreach $a \in [l, r]$ **do**
  if $([inf, sup] \cap \mathcal{I}(\min_{x\in[a,r]} e)) = \varnothing$ **then break**
  $\langle ok, res\rangle \leftarrow eval(e|_{x:=a}, [inf, sup])$
  if ok **then**
    $\langle solution\_found, val\rangle \leftarrow \langle true, res\rangle$
    $[inf, sup] \leftarrow [inf, sup] \cap [-\infty, res]$
return $\langle solution\_found, val\rangle$

Notez que la recherche peut être interrompue à n'importe-quelle itération si l'évaluation intervalle nous apprend qu'aucune solution ne sera trouvée dans $[a, r]$. Des idées similaires s'appliquent au cas des sommes itérées :

function $eval(\sum_{x\in[l,r]} e, [inf, sup])$

$sum \leftarrow 0$
foreach $a \in [l, r]$ **do**
  $[inf', sup'] \leftarrow [inf, sup] - sum - \mathtt{I}(\sum_{x\in[a+1,r]} e)$
  if $[inf', sup'] = \varnothing$ **then return** $\langle false, \_\rangle$
  $\langle ok, res\rangle \leftarrow eval(e|_{x:=a}, [inf', sup'])$
  if $\neg ok$ **then return** $\langle false, \_\rangle$
  $sum \leftarrow sum + res$
return $\langle true, sum\rangle$

5.2 Integration dans un Résolveur

Pour intégrer les EIs dans un résolveur CP classique, la principale question est de savoir comment les mécanismes de recherche arborescente des deux types de variables interagissent : nous avons d'un coté les variables CP, prises en compte par l'algorithme de recherche classique, et de l'autre les variables introduites par les EIs, et qui sont prises en compte par nos nouveaux algorithmes de *Branch & Bound*. Notre solution est pour le moment d'énumérer d'abord sur les variables CP classiques . Quand nous considérons une contrainte

[5] Cet intervalle devrait en théorie être ouvert, *i.e.,* $[0, 4.7[$, mais ceci est souvent délicat a implémenter sur les flottants. Notez les similarités de notre algorithme à la fois avec les algorithmes de [9] et avec les techniques de *Branch & Bound* par intervalles pour les contraintes continues.

itérée, toutes ses variables libres sont alors instanciées (d'où notre supposition initiale pour les algorithmes précédents), et nous pouvons utiliser le Branch & Bound pour vérifier que les contraintes sont satisfaites. De meilleures heuristiques de recherche peuvent très probablement être définies, mais la question de l'ordre d'énumération dans les EIs est une problématique de recherche *per se*, qui est aussi non-triviale que pour les contraintes quantifiées ou stochastiques.

Nous avons considéré des EIs basiques dans les algorithmes pour des raisons de simplicité de présentation, mais il est facile d'adapter les algorithmes a des EIs contraintes : dans les boucles il suffit d'ignorer les valeurs qui ne satisfont pas ces contraintes. Par ailleurs, la propagation (classique) entre les variables liées aux connecteurs itérés doivent être réactivées régulièrement.

6 Discussion

6.1 Littérature Existante

Un grand nombre de travaux liés a l'incertain reposent d'une certaine manière sur des expressions itérées mais "dissimulent" celles-ci derrière un cadre non-classique ; voir par exemple [7]. Notre approche est, à l'inverse, de donner à l'utilisateur la possibilité de manipuler ces expressions directement. Les raisons de ce choix sont essentiellement pragmatiques : notre but dans ce travail n'était pas d'exprimer toutes les formes d'incertain au sein d'un cadre général, mais d'outiller CP avec des extensions minimales permettant de prendre en compte certaines formes d'incertain (*espérance* de certaines variables aléatoires, en particulier). Nous voyons les avantages de notre travail comme complémentaires a celles de ces autres approches : (1) les expressions itérées sont une construction mathématique comprise et utilisée par tous et ne s'exposent donc pas au risque d'être perçues comme "une construction de plus dont l'emploi est réservé aux seuls experts" ; (2) leur intégration dans un résolveur est relativement naturelle, par opposition au reproche souvent fait aux travaux sur l'incertain qui est que leur intégration demanderait de revoir le cœur même du résolveur ; (3) étant une extension du cadre CP, les EIs permettent de construire sur l'existant sans fondamentalement changer d'approche et d'ajouter un traitement de l'incertain a certaines applications qui sont déjà déployées mais ignorent les aspects incertains du problème, par manque d'outil.

Les travaux les plus directement liés aux EIs sont ceux sur les CSP stochastiques (SCSP) [9, 8, 1]. En SCSP, nous optimisons essentiellement une fonction f dont les paramètres sont une séquence de variables $X_1 \dots X_n$ incluant un sous-ensemble S de variables aléatoires. Les CSP stochastiques considèrent des expressions de forme :

$$\bigoplus_{x_1 \in D_1}^{1} \cdots \bigoplus_{x_n \in D_n}^{n} \left(f(a_1 \dots a_n) \cdot \prod_{i \in S} \Pr(Y_i = a_i) \right) \quad (21)$$

Oú $\bigoplus^i$ est $\sum$ si $X_i \in S$ et min dans le cas contraire. Les SCSP représentent donc une forme particulière d'expressions itérées dont la structure est *séquentielle*. Lors de nos investigations des SCSP nous les avons trouvés difficiles a utiliser pour nos besoins, pour plusieurs raisons : (1) comme remarqué dans [4], les CSP stochastiques ne peuvent pas être utilisés pour des workflows non-séquentiels (pas de "branchement"), alors que la traduction de l'ensemble des constructions de base des workflows en EIs est relativement naturelle ; (2) pour permettre de gérer des instances réelles, il manque à SCSP une notion de "quantificateurs restreints" qui, dans notre approche, correspondent a une construction naturelle : les EIs contraintes ; (3) l'approche SCSP est complexe et les algorithmes de *Branch & Bound* disponibles a l'heure actuelle [1] sont réservés au cas particulier ou l'objectif représente l'espérance de *satisfaction* des contraintes, et où les probabilités sont indépendantes. Pour faire face a ces problèmes, nous avons besoin d'étendre l'approche SCSP et ses algorithmes ; en y travaillant nous avons remarqué que nous étions en permanence en train de raisonner sur des $\sum$s et min, et que le cœur du problème se trouve dans ces constructions.

6.2 Conclusion

Notre étude des expressions itérées révèle que ces constructions mathématiques de base, qui sont absentes des outils CP de manière quelque-peu surprenantes, permettent d'exprimer des problèmes complexes ayant des applications importantes, notamment des problèmes présentant de l'incertain. Nous avons proposé un algorithme basique et général pour la propagation d'EIs, des propagateurs améliorés pour un ensemble de cas particuliers intéressants, et une méthode de Branch & Bound. Ces contributions montrent comment intégrer les EIs dans un résolveur CP classique.

La prochaine étape est d'évaluer les performances de l'approche et d'améliorer son passage à l'échelle. Dans les applications que nous avons considérées (workflows) il est en général possible d'éviter des temps de calcul trop longs en limitant la profondeur de raisonnement (*e.g.,* si besoin on peut se contenter de raisonner "deux activités d'avance" sur le futur). Plus généralement, il serait intéressant d'étudier des moyens

d'utiliser des méthodes de recherche incomplète pour la résolution d'EIs.

Références

[1] T. Balafoutis and K. Stergiou. Algorithms for stochastic CSP. In *Proc. of Int. Conf. on Principles and Practice of Constraint Programming (CP)*, pages 44–58. Springer, 2006.

[2] M. Benedetti, A. Lallouet, and J. Vautard. QCSP made practical by virtue of restricted quantification. In *Proc. of Int. Joint. Conf. on Artificial Intelligence (IJCAI)*, pages 38–43. Morgan Kaufmann, 2007.

[3] F. Benhamou, D. A. McAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In *Proc. of Int. Logic Programming Symposium*, pages 124–138. MIT Press, 1984.

[4] L. Bordeaux and H. Samulowitz. On the stochastic constraint satisfaction framework. In *Proc. of ACM Int. Symp. on Applied Computing (SAC)*, pages 316–320. ACM, 2007.

[5] K. N. Brown, J. Little, and E. C. Freuder. Adversarial constraint satisfaction by game-tree search. In *Proc. of Euro. Conf. on Artificial Intelligence (ECAI)*, pages 151–155. IOS Press, 2004.

[6] G. Pesant. Counting solutions of CSPs : A structural approach. In *Proc. of Int. Joint. Conf. on Artificial Intelligence (IJCAI)*, pages 260–265. Morgan Kaufmann, 2005.

[7] C. Pralet, T. Schiex, and G. Verfaillie. Decomposition of multi-operator queries on semiring-based graphical models. In *Proc. of Int. Conf. on Principles and Practice of Constraint Programming (CP)*, pages 437–452. Springer, 2006.

[8] A. Tarim, S. Manandhar, and T. Walsh. Stochastic constraint programming : A scenario-based approach. *Constraints*, 11(1) :53–80, 2006.

[9] T. Walsh. Stochastic constraint programming. In *Proc. of Euro. Conf. on Artificial Intelligence (ECAI)*, pages 111–115. John Wiley and Sons, 2002.