Contraintes sur les Domaines Continus

—Notes de cours (brouillon)—

Michel RUEHER

Décembre 2003

Table des matières

1 Arithmétique des Intervalles 3
1.1 Notations . . . 3
1.2 Bases du calcul d'intervalles . . . 3

2 Propriétés 5
2.1 CSP "numérique" . . . 5
2.2 Sémantique d'un CSP . . . 6
2.3 Approximation d'un CSP . . . 7
2.4 Algorithme standard de narrowing . . . 7
2.5 Consistances partielles . . . 7

3 2–B–Consistance 8
3.1 Consistance d'arc versus 2–B–Consistance . . . 8
3.2 filtrage par 2–B–Consistance . . . 9
3.3 Calcul de la 2–B–Consistance . . . 9
3.4 Algorithme de calcul de la 2–B–Consistance . . . 11

4 2–B(w)–Consistance 12
4.1 Arrêt prématuré de l'algorithme de propagation . . . 12
4.2 Problèmes de la 2B(w)–Consistance . . . 13

5 3B–Consistance 15
5.1 3B–Consistance[18] . . . 15

6 La Box–Consistance 18
6.1 Définition et propriétés de la Box–Consistance . . . 18
6.2 Calcul de la Box–Consistance . . . 20

7 Bound–Consistance 21

8 Efficacité de la Box–Consistance et de la 2B–Consistance 21
8.1 Coûts de calcul 22
8.2 Techniques d'accélération des algorithmes de filtrage 23

9 Utilisation des techniques de propagation d'intervalle sur les domaines finis et sur les booléens 24
9.1 La consistance d'intervalles sur les domaines finis 24
9.2 Filtrage de système de contraintes booléens 25

1 Arithmétique des Intervalles

1.1 Notations

Nous utilisons les notations suivantes, éventuellement indicées :
- x, y, z représentent des variables définies sur les réels ; X, Y, Z représentent des variables définies sur les intervalles ;
- u, v représentent des constantes dans $\mathcal{R}$;
- f, g représentent des fonctions définies sur les réels ; F, G représentent des fonctions définies sur les intervalles ;
- c correspond à une contrainte définie sur les réels ; C correspond à une relation définie sur les intervalles ;
- $\Phi_{cstc}(P)$ est la fermeture (ou le filtrage) par la consistance *cstc* (où *cstc* est $2B$, Box, $3B$, $Bound$) de P .

$\mathcal{R}^\infty = \mathcal{R} \cup \{-\infty, +\infty\}$ est l'ensemble des réels étendu aux symboles des infinis. $\overline{\mathbb{F}}$ est un sous-ensemble fini de $\mathcal{R}^\infty$ contenant en particulier $\{-\infty, +\infty\}$. a, b représentent des constantes dans $\overline{\mathbb{F}}$. Concrètement, $\overline{\mathbb{F}}$ correspond à un ensemble de nombres flottants utilisés dans une implémentation machine. a^+ (resp. a^-) correspond au plus petit (resp. plus grand) nombre de $\overline{\mathbb{F}}$ strictement plus grand (resp. plus petit) que a.

1.2 Bases du calcul d'intervalles

Définition 1.1 (Intervalle). Un intervalle $[a, b]$ avec $a, b \in \overline{\mathbb{F}}$est l'ensemble de nombres réels $\{r \in \mathcal{R} \mid a \leq r \leq b\}$.

Soit r un nombre réel. $\tilde{r}$ correspond au plus petit (w.r.t. inclusion) intervalle de $\overline{\mathbb{F}}$ contenant r. $\mathcal{I}$ correspond à l'ensemble de tous les intervalles et est partiellement ordonné par l'inclusion ensembliste. $\mathcal{U}(\mathcal{I})$ correspond à l'ensemble de toutes les unions d'intervalles.

Commentaire 1.1 (Incidence de la précision des calculs).
Le terme plus petit (w.r.t. inclusion) sous-ensemble doit être interprêté en fonction de la précision des calculs lors des opérations en virgule flottante. Nous supposerons ici que :

1. tous les résultats des calculs sont arrondis vers l'extérieur (i.e., vers $-\infty$ lors du calcul de la borne inférieure d'un intervalle et vers $+\infty$ lors du calcul de la borne supérieure d'un intervalle) ;
2. l'erreur maximale lors du calcul d'une borne de l'intervalle associé à une variable du système de contraintes intial est toujours inférieure à un flottant.

La deuxième hypothèse peut nécessiter de recourir à des "grands flottants" (flottants en précision arbitraire avec une mantisse de taille variable)[5] et peut donc paraître un peu forte. Elle est toutefois requise pour une comparaison précise des filtrages effectués à l'aide des différentes techniques de

consistance partielle. En effet, si les résultats des opérations élémentaires qui ne sont pas des flottants sont arrondis aux flottants précédents (ou suivants), les algorithmes de filtrage basés sur une consistance partielle ne calculent plus un point fixe unique[18].

Définition 1.2 (Extension ensembliste). Soit S un sous ensemble de $\mathcal{R}$. l'approximation de S —notée $\Box S$— est le plus petit intervalle I tel que $S \subseteq I$.

Définition 1.3 (Extension aux intervalles [24, 14]).
• Une fonction sur les intervalles $F : \mathcal{I}^n \rightarrow \mathcal{I}$ est une extension aux intervalles de $f : \mathcal{R}^n \rightarrow \mathcal{R}$ ssi :

$$\forall I_1, \ldots, I_n \in \mathcal{I} : r_1 \in I_1, \ldots, r_n \in I_n \Rightarrow f(r_1, \ldots, r_n) \in F(I_1, \ldots, I_n).$$

• Une relation sur les intervalles $C : \mathcal{I}^n \rightarrow \mathcal{B}ool$ est une extension aux intervalles de la relation $c : \mathcal{R}^n \rightarrow \mathcal{B}ool$ ssi :

$$\forall I_1, \ldots, I_n \in \mathcal{I} : r_1 \in I_1, \ldots, r_n \in I_n \Rightarrow [c(r_1, \ldots, r_n) \Rightarrow C(I_1, \ldots, I_n)]$$

Par exemple, la relation sur les intervalles $\doteq$ definie comme $I_1 \doteq I_2 \Leftrightarrow (I_1 \cap I_2) \neq \emptyset$ est une extension aux intervalles de la relation d'égalité sur les réels.

Définition 1.4 (Extension naturelle aux intervalles [24, 25]).
Une fonction sur les intervalles $F : \mathcal{I}^n \rightarrow \mathcal{I}$ est une extension naturelle aux intervalles de $f : \mathcal{R}^n \rightarrow \mathcal{R}$ ssi F est l'extension aux intervalles de f obtenue en remplaçant dans f, chaque constante k par son extension naturelle aux intervalles $\tilde{k}$, chaque variable par une variable sur les intervalles et chaque opération arithmétique par son extension optimale aux intervalles [24]. De manière informelle, on peut définir l'extension optimale comme le plus petit intervalle qui conserve l'ensemble des solutions d'une fonction continue.

Les extensions optimales aux intervalles ont été introduites par Moore[24] pour les opérations de base sur les intervalles.

Par exemple, soit $\odot$ un opérateur binaire parmi $\{+, -, *, /\}$ et $[a, b] \odot [c, d] = \{x \odot y$ tel que $a \leq x \leq b$ et $c \leq y \leq d\}$ alors l'extension optimale aux intervalles pour les qautres opérations de base est définie par :

- $[a, b] \ominus [c, d] = [a - d, b - c]$
- $[a, b] \oplus [c, d] = [a + c, b + d]$
- $[a, b] \otimes [c, d] = [min(ac, ad, bc, bd), max(ac, ad, bc, bd)]$
- $[a, b] \oslash [c, d] = [min(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}), max(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d})]$ if $0 \notin [c, d]$

Dans la suite on notera $\oplus, \ominus, \otimes, \oslash$ les extensions optimales aux intervalles de $+, -, \times, /$.

Les extensions optimales aux intervalles peuvent être définies de manière similaire pour quasiment toutes les fonctions élémentaires [25].

Exemple 1.1. Soit $f(x) = x + x \times y + 3$ une fonction sur les réels. Son extension naturelle aux intervalles est définie par $X \oplus X \otimes Y \oplus \tilde{3}$.

Il est important de remarquer que la forme syntaxique d'une fonction a une très forte influence sur la précision de l'extension naturelle aux intervalles qui lui sera associée.

Exemple 1.2. [31] Soit les fonctions sur les réels définies par :
- $f_1(x) : x^2 - x$;
- $f_2(x) : x(x-1)$.

L'extension naturelle aux intervalles nous donne le résultat suivant :
- $F_1([0,5]) = [-5,25]$;
- $F_2([0,5]) = [-5,20]$.

alors que la valeur de l'extension optimale aux intervalles de ces deux fonctions sur l'intervalle $[0,5]$ correspond à l'intervalle $[-0.25,20]$: le domaine de variation de f_1 et f_2 est $[-0.25,20]$ lorsque x varie entre $[0,5]$ (la dérivée de ces deux fonctions s'annullent pour $x = 0.5$).

2 Propriétés fondamentales de l'arithmétique des intervalles

Les opérateurs $\{+,*\}$ sont commutatifs et associatifs mais la distributivité n'est pas vérifiée.

Proposition 1. [24] Soit : $F : \mathcal{I}^n \to \mathcal{I}$ l'extension naturelle aux intervalles de : $f : \mathcal{R}^n \to \mathcal{R}$ et soit $f_{sol} = \square\{f(r_1,\ldots,r_n) \mid r_1 \in I_1,\ldots,r_n \in I_n\}$. Si chaque r_i a une seule occurrence dans f alors $f_{sol} = F(I_1,\ldots,I_n)$ sinon $f_{sol} \subseteq F(I_1,\ldots,I_n)$.

Ce résultat peut aisément être étendu [6] aux relations k-aires sur $\mathcal{R}^n$:

Proposition 2. Soit $C : \mathcal{I}^n \to \mathcal{B}ool$ l'extension naturelle aux intervalles d'une équation $c : \mathcal{R}^n \to \mathcal{B}ool$. Si chaque x_i a une seule occurrence dans c, alors : $C(I_1,\ldots,I_n) \Leftrightarrow (\exists x_1 \in D_1,\ldots,\exists x_n \in D_n \mid c(x_1,\ldots,x_n))$.

Exemple 2.1. Voici quelques exemples qui illustrent les limites de la sous-distributivité :

a) $[1,2] * ([1,2] - [1,2]) = [-2,2]$
$[1,2] * [1,2] - [1,2] * [1,2] = [-3,3]$

b) $f_1 : \mathcal{R} \to \mathcal{R}$ $\quad$ $F_1 : I \to I$
$x \to x - x$ $\quad$ $i \to i - i$
Si $I_0 = [10,20]$ $\quad$ $F_1(I_0) = [-10,10]$

2.1 Système de contraintes sur les intervalles

Définition 2.1. Un CSP sur les domaines continus est un triplet :
- $(\mathcal{X},\mathcal{D},\mathcal{C})$ où $\mathcal{X} = \{x_1,\ldots,x_n\}$ est un ensemble de variables ;
- $\mathcal{D} = \{D_{x_1},\ldots,D_{x_n}\}$ est un ensemble de domaines (D_{x_i} est le domaine contenant toutes les valeurs potentielles de la variable x_i) ;

– $\mathcal{C} = \{c_1, \ldots, c_m\}$ est un ensemble de contraintes.

On note $Var(C)$ le sous ensemble de $\mathcal{X}$ des variables de C.

Exemple 2.2. Voici quelques exemples de CSP sur les domaines continus :

a) $P_0 = (\mathcal{V}, \{D_U, D_R, D_I\}, \{U = R \times I\})$
avec $D_R = [10.5, 11], D_I = [1, 2], D_U = (-\infty, +\infty)$

b) $P_1 = (V, D_1, C)$ $P_2 = (V, D_2, C)$ $P_3 = (V, D_3, C)$
avec :
$V = \{x, y\}$
$D_1 = \{D_x = [0, 100], \quad D_y = [0, 100]\}$
$D_2 = \{D_x =]0, 2], \quad D_y = [0, 2[\}$
$D_3 = \{D_x = [0.5, 1], \quad D_y = [1, 1.5]\}$
$C = \{x + y = 2,\ y \leq x + 1,\ y \geq 1 + \ln x\}$

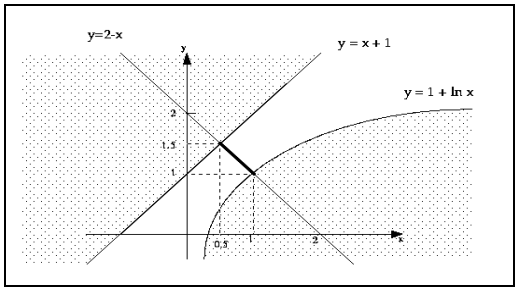

Fig. 1 – Contraintes des CSP P_1, P_2 et P_3

On notera que les CSP P_1, P_2 et P_3 ont le même ensemble de solutions (cf. fig. 1).

$P_\emptyset$ représente la classe des CSP vides, i.e., l'ensemble des CSP dont au moins un domaine est vide. $\mathcal{D}' \subseteq \mathcal{D}$ signifie que $D'_{x_i} \subseteq D_{x_i}$ pour tout $i \in 1..n$. On considère qu'un CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est plus petit qu'un CSP $P' = (\mathcal{X}, \mathcal{D}', \mathcal{C})$ si $\mathcal{D}' \subseteq \mathcal{D}$. On note $P \prec P'$ cette relation d'ordre. Par convention $P_\emptyset$ est le plus petit des CSP.

2.2 Sémantique d'un CSP

Soit le CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ avec $\mathcal{X} = \{x_1, \ldots, x_n\}$, $\mathcal{D} = \{D_1, \ldots, D_n\}$ et $\mathcal{C} = \{c_1, \ldots, c_m\}$ où chaque contrainte est d'arité n. L'ensemble des solutions du CSP P est défini par la relation $\cap_{i=1}^{i=m} c_i \cap (D_{x_{j_1}} \times \cdots \times D_{x_{j_k}})$.

2.3 Approximation d'un CSP

Comme il est en général impossible de calculer toutes les solutions d'un CSP, on cherche à calculer une approximation de cet ensemble de solutions.

Définition 2.2 (domaine d'approximation). [3] Un domaine d'approximation A sur $\mathcal{R}$ est un sous-ensemble des parties de $\mathcal{R}$ clos par intersection, muni d'une relation d'ordre bien fondée, et tel que $\mathcal{R} \in A$.

$\mathcal{I}$ correspond à l'ensemble des intervalles dont les bornes appartiennent à $\overline{I\!F}$, un sous ensemble fini de $\mathcal{R}$. $\mathcal{I}$ est partiellement ordonné par l'inclusion ensembliste et constitue donc un domaine d'approximation de $\mathcal{R}$. Cet ensemble est un treillis complet dont la borne inférieure est définie par l'intersection ensembliste et la borne supérieure par l'union ensembliste.

Définition 2.3 (opérateur de narrowing). Soit A un domaine d'approximation sur $\mathcal{R}$, ρ une relation n-aire sur $\mathcal{R}$. La fonction $N : A^n \rightarrow A^n$ est un opérateur de narrowing pour la relation ρ ssi pour tout $u, v \in A^n$ on a :

1. $N(u) \subset u$,
2. $u \cap \rho \subset N(u)$,
3. $u \subset v \Rightarrow N(u) \subset N(v)$

La première propriété correspond à la contractance, la deuxième à la correction et la troisième à la monotonie.

2.4 Algorithme standard de narrowing

Un algorithme standard d'approximation d'un CSP est défini dans la figure 2. Cet algorithme est dérivé de l'algorithme AC3 utilisé pour caluler la consistance d'arc sur les domaines finis. L'approximation qui sera effectivement calculée est définie par la consistance locale associée à l'opérateur de `narrowing`.

2.5 Consistances partielles

Les consistances partielles utilisées au niveau des CSP sur les domaines continus se divisent en deux grandes catégories :

- les consistances strictement locales, i.e., qui ne travaillent que sur une seule contrainte ;
- les consistances partielles qui ne sont pas strictement locales ,i.e., qui vérifient certaines propriétés sur l'ensemble (ou sur un sous-ensemble) du système de contraintes locales

```
1 IN–1 (in C, inout D)
2    Q ← {< x_i, C_j > | C_j ∈ C and x_i ∈ Var(C_j)}
3    while Q ≠ ∅
4          extract < x_i, C_j > from Q
5          D' ← narrowing(D, x_i, C_j)
6          if D' ≠ D then
7                D ← D'
8                Q ← Q ∪ {< x_l, C_k > | C_k ≠ C_j ∧ (x_l, x_i) ∈ Var(C_k)}
10         endif
11   endwhile
```

Fig. 2 – Algorithme standard d'approximation d'un CSP

3 2–B–Consistance

Intuitivement, la 2–B–Consistance[26, 27, 18, 2, 28]) est une approximation de la consistance d'arc[22] et est donc une consistance strictement locale. Schématiquement, la contrainte c est 2B–Consistante si pour toute variable x de domaine $D_x = [a, b]$ il existe des valeurs dans les domaines de toutes les autres variables qui satisfont c lorsque x est instancié avec a et lorsque x est instancié avec b. En d'autres termes, si c peut s'écrire sous la forme $f(x) = 0$, alors a et b sont respectivement le plus petit et le plus grand zéro de $f(x)$ dans l'espace délimité par $D_{x_1} \times \ldots \times D_{x_n}$.

Définition 3.1 (2B–Consistance). Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP et $c \in \mathcal{C}$ une contrainte k-aire sur les variables $(x_1, \ldots, x_k)$. La contrainte c est 2B–Consistante ssi :
$\forall i, D_{x_i} = \Box\{v_i \in D_{x_i} \mid \exists v_1 \in D_{x_1}, \ldots, \exists v_{i-1} \in D_{x_i-1}, \exists v_{i+1} \in D_{x_i+1}, \ldots,$
$\exists v_k \in D_{x_k}$ tel que $c(v_1, \ldots, v_{i-1}, x_i, v_{i+1} \ldots, v_k)$ est vérifiée $\}$.
Un CSP est 2B–Consistant ssi toutes ses contraintes sont 2B–Consistantes.

3.1 Consistance d'arc versus 2–B–Consistance

La 2B–Consistance est une consistance plus faible que la consistance d'arc :

- La consistance d'arc impose une condition sur tous les éléments des domaines ;
- La 2–B–Consistance impose cette même condition aux seules bornes de l'intervalle qui contient les domaines.

Exemple 3.1. $P = \{\{x, y\}, \{D_x, D_y\}, \{x = y^2\}\}$ avec $D_x = [1, 4]$, $D_y = [-2, +2]$
P est 2–B–Consistant, mais pas consistant d'arc car la valeur 0 de D_y n'a pas de support dans D_x.

La 2–B–Consistance ne garantit naturellement pas l'existence d'une solution.

Exemple 3.2. $P' = \{\{x, y\}, \{D_x, D_y\}, \{x^2 = 4, y^2 = 9, x * y = 1\}\}$
avec $D_x =]-2, 2[,\ D_y = [-3, +3]$
P' est 2–B–Consistant, mais n'a pas de solution.

La propriété suivante explicite la relation entre la 2B–Consistance et l'extension aux intervalles d'une contrainte.

Proposition 3. Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP tel qu'aucune contrainte de $\mathcal{C}$ ne contienne des occurrences multiples des variables de $\mathcal{X}$, soit $c \in \mathcal{C}$ une contrainte d'arité k définie sur $(x_1, \ldots, x_k)$. c est 2–B–Consistant ssi pour tout x_i de $\{x_1, \ldots, x_k\}$ tel que $D_{x_i} = [a, b]$ les relations ci-dessous sont vérifiées :
- $C(D_{x_1}, \ldots, D_{x_{i-1}}, [a, a^+), D_{x_{i+1}}, \ldots, D_{x_k})$,
- $C(D_{x_1}, \ldots, D_{x_{i-1}}, (b^-, b], D_{x_{i+1}}, \ldots, D_{x_k})$.

où $[a, a^+)$ et $(b^-, b]$ sont des intervalles semi-ouverts.

3.2 filtrage par 2–B–Consistance

Définition 3.2 (filtrage par 2B–Consistance). Un filtrage par 2–B–Consistance d'un CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est un CSP $P' = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ tel que :

1. $P' \equiv P$, i.e. P et P' ont les mêmes solutions ;
2. P' est 2–B–Consistant,
3. $P' \preceq P$,
4. $\mathcal{D}' \subset \mathcal{D}$ et les domaines de $\mathcal{D}'$ sont les plus grands domaines tel que P' soit un filtrage par 2–B–Consistance de P.

Le filtrage par 2–B–Consistance existe toujours et est unique.

On note $P' = \Phi_{2B}(\mathrm{P})$. Ainsi, dans l'exemple 1 : $P_2 = \Phi_{2B}(P_1)$.

3.3 Calcul de la 2–B–Consistance

Le calcul de la 2–B–Consistance repose sur l'approximation des fonctions de projection utilisées pour réduire le domaine des variables dans la fonction `narrowing`$(\mathcal{D}, x, C)$ de `IN-1` (cf. figure 2).

Soit c une contrainte k-aire sur $(x_1, \ldots, x_k)$, et $< I_1, \ldots, I_k > \in \mathcal{I}^k$ un ensemble de domaines, $\pi_i(c, I_1 \times \cdots \times I_k)$ dénote la projection sur x_i de l'ensemble des solutions de c dans l'espace délimité par $I_1 \times \cdots \times I_k$.

Définition 3.3 (projection d'une contrainte).
$\pi_i(c, I_1 \times \cdots \times I_k) : (\mathcal{C}, \mathcal{I}^k) \rightarrow \mathcal{U}(\mathcal{I})$ est la projection de c sur x_i ssi :
$\pi_i(c, I_1 \times \cdots \times I_k) = \{v_i \in I_i \mid \exists < v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k > \in I_1 \times \cdots \times I_{i-1} \times I_{i+1}, \ldots \times I_k$ tel que $c(v_1, \ldots, v_i, \ldots, v_k)$ est vrai $\}$

La projection d'une contrainte sur une variable n'est en général pas un intervalle mais une union d'intervalles. Pour éviter de devoir gérer les unions d'intervalles on approxime cette projection par un intervalle.

Définition 3.4 (approximation de la projection). $AP_i(c, I_1 \times \cdots \times I_k) : (\mathcal{C}, \mathcal{I}^k) \rightarrow \mathcal{I}$ est une approximation de $\pi_i(c, I_1 \times \cdots \times I_k)$ ssi
$AP_i(c, I_1 \times \cdots \times I_k) = \square\ \pi_i(c, I_1 \times \cdots \times I_k)$
$= [Min\ \pi_i(c, I_1 \times \cdots \times I_k), Max\ \pi_i(c, I_1 \times \cdots \times I_k)]$.
En d'autres termes, $AP_i(c, I_1 \times \cdots \times I_k)$ est le plus petit intervalle contenant la projection $\pi_i(c, I_1 \times \cdots \times I_k)$.

Il resulte de ces définitions qu'une contrainte c est 2B–Consistante sur $< I_1, \ldots, I_k >$ si pour tout i dans $\{1, \ldots, k\}$, $I_i = AP_i(c, I_1 \times \cdots \times I_k)$.

Le calcul de AP_i est immédiat pour les fonctions monotones : il suffit d'évaluer la fonction de projection pour les bornes de l'intervalle associé à la variable x_i. Par contre, un des problèmes majeurs de la 2–B–Consistance est que AP_i ne peut pas être calculée directement dans le cas général où il est difficile de définir les fonctions Min et Max ; en particulier si une variable à des occurrences multiples. La solution consiste alors à décomposer les contraintes en fonctions primitives pour lesquelles il est facile d'identifier les parties monotones.

Exemple 3.3 (évaluation des fonctions de projection). Voici deux exemples d'évaluation de fonctions de projection :

(a) Soit $x = y + z^2$ et $D_x = [0, 6], D_y = [5, 9], D_z = [-2, 4]$.
La fonction de projection pour y est $D'_y \leftarrow (D_x - D_z^2) \cap D_y$.
D'ou : $D'_y = ([0, 6] - [0, 16]) \cap [5, 9] = [5, 6]$.
On remarquera que pour calculer D_z^2 on a utilisé l'extension optimale aux intervalles de la fonction puissance et non le produit.

(b) Soit $x + y^2$ et $D'_y \leftarrow \sqrt{D_x} \cap D_y$
L'évaluation de la fonction de projection demande dans ce cas une analyse des parties monotones :
Si $D_x = [4, 9]$ et $D_y = [-1, 5]$ alors D'_y vaut $[2, 3]$ et non $[-1, 3]$ car aucune des valeurs entre -1 et $\sqrt{D_x}$ n'a de support dans D_x. Par contre, si $D_x = [4, 9]$ et $D_y = [-2, 5]$ alors D'_y vaut $[-2, 3]$ car -2 a pour support $-\sqrt{4}$.

En général on décompose chaque contrainte en contraintes binaires ou ternaires par l'introduction de nouvelles variables. D'un point de vue théorique, il suffit toutefois d'introduire des nouvelles variables pour éliminer les occurrences multiples des variables.

La décomposition d'un système de contraintes $\mathcal{C}$ en un système de contraintes primitives $decomp(\mathcal{C})$ ne change pas la sémantique : $\mathcal{C}$ et $decomp(\mathcal{C})$ ont les mêmes solutions.Toutefois, pour une propriété locale comme la 2B–Consistance, la portée des vérifications effectuées par la 2B–Consistante est réduite

par la décomposition. Ceci n'est pas grave, si aucune variable n'a d'occurrences multiples : dans ce cas la propagation garantit que les filtrages par 2B–Consistance de $\mathcal{C}$ et de $decomp(\mathcal{C})$ sont identiques. Par contre, si une variable x possède des occurrences multiples dans une contrainte $c \in \mathcal{C}$, alors les différentes occurrences de x dans $decomp(c)$ pourront varier indépendamment les unes des autres et le filtrage par 2B–Consistance de $decomp(\mathcal{C})$ sera plus faible que celui de $\mathcal{C}$.

Exemple 3.4 (décomposition du système de contraintes).
Soit $c \ : x_1 + x_2 - x_1 = 0$ une contrainte $D_{x_1} = [-1, 1]$, $D_{x_2} = [0, 1]$ les domaines de x_1 et x_2. Puisque x_1 apparait deux fois dans c, sa deuxième occurrence sera remplacée par une nouvelle variable x_3 .
On a donc $decomp(c) = \{x_1 + x_2 - x_3 = 0, x_1 = x_3\}$.
Dans ce nouveau système de contraintes, chaque fonction de projection peut être évaluée aisément à l'aide des opérations du calcul d'intervalle. Par exemple :
$AP_1(x_1 + x_2 - x_3 = 0, D_{x_1}, D_{x_2}, D_{x_3})$ est $D_{x_1} \cap (D_{x_3} \ominus D_{x_2})$.

Toutefois la décomposition accentue les limites due à la localité de la 2–B–Consistance : la contrainte initiale n'est pas 2–B–Consistante (aucune valeur x_1 ne permet de satisfaire la contrainte lorsque $x_2 = 1$) alors que le système décomposé est 2–B–Consistant (les valeurs $x_1 = -1$ et $x_3 = 0$ satisfont $x_1 + x_2 - x_3 = 0$ lorsque $x_2 = 1$).

3.4 Algorithme de calcul de la 2–B–Consistance

L'algorithme de calcul de la 2–B–Consistance résulte directement de l'instanciation de la fonction `narrowing` par le calcul des fonctions extremum (cf. figure 3) dans l'algorithme `IN-1`.

```
1 function narrow (D, x_i, C_j) : set of domains
2     m ← Min_{x_i}(C, D_{x_i})
3     M ← Max_{x_i}(C, D_{x_i})
4     return  D[D_{x_i} ← [m, M]]
```

Fig. 3 – Fonction `narrowing` pour le calcul de la 2–B–Consistance

Exemple 3.5 (Mise en oeuvre du filtrage par 2–B–Consistance). Soit $c_1 : x + y = 1, \quad c_2 : y = 2 * z, \quad D_x = [-4, 5], D_y = [3, +\infty), D_z = [1, 2]$
La file des couples <contrainte,variable> à traiter est $Q = \{a :< c_1, x >, b :< c_1, y >, c :< c_2, y >, d :< c_2, z >\}$

1. Sélection de a, $D_x \leftarrow [-4, -2]$.
   Aucun élément n'est ajouté dans Q.

2. Sélection de b, $D_y \leftarrow [3,5]$.
   Aucun élément n'est ajouté dans Q.
3. Sélection de c, $D_y \leftarrow [3,4]$.
   $Q = \{d :< c_2, z >, a :< c_1, x >\}$
4. Sélection de d, $D_z \leftarrow [1.5,2]$.
   Aucun élément n'est ajouté dans Q.
5. Sélection de a, $D_x \leftarrow [-3,-2]$.
   Aucun élément n'est ajouté dans Q.
6. $Q = \emptyset$

Remarque : comme l'exemple est linéaire le même résultat aurait pu être obtenu en utilisant l'algorithme du simplexe, d'abord pour déterminer la satisfiabilité du système de contraintes, ensuite pour calculer l'optimum de chaque borne de chaque variable.

4 2–B(w)–Consistance

4.1 Arrêt prématuré de l'algorithme de propagation

Lors de convergences asymptotiques il est en général préférable d'arrêter la propagation avant d'atteindre le point fixe. En effet, il n'est pas réaliste dans ce cas de vouloir réduire tous les intervalles jusqu'à ce qu'aucun élément de $\overline{\mathbb{F}}$ (e.g., aucun flottant) ne puisse être enlevé.

Exemple 4.1. Soit :
$X = 2 \times Y$
$Y = X$
$D_X = [-10,10], D_Y = [-10,10]$
La 2B-consistance va opérer les réductions suivantes :

$D_Y = [-5,5]$	$D_X = [-5,5]$
$D_Y = [-2.5,2.5,$	$D_X = [-2.5,2.5]$
$D_Y = [-1.25,1.25]$	$D_X = [-1.25,1.25]$
$D_Y = [-0.625,0.625]$	$D_X = [-0.625,0.625]$
.....	

Pour des questions de performance, il est clair sur un tel exemple qu'il est préférable d'arrêter la propagation avant d'atteindre le point fixe.

On introduit de ce fait une notion de "largeur" de borne (cf. figure 4) et on se contente de rechercher une solution dans cette borne large. On appelle 2B(w)–Consistance la consistance obtenue lors d'un tel arrêt prématuré de l'algorithme de filtrage.

Formellement, la 2B(w)–Consistance peut être définie de la manière suivante :

Définition 4.1. Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP, $x \in \mathcal{X}$, $D_x = [a, b]$, w un entier positif. D_x est 2–B(w)–Consistant si pour toute contrainte $c(x, x_1, \ldots, x_k)$ de $\mathcal{C}$ si les relations suivantes sont vérifiées :
1) $\exists v \in [a, a^{+w}),\ \exists v_1, \ldots, v_k \in D_{x_1} \times \cdots \times D_{x_k} \mid c(v, v_1, \ldots, v_k)$
2) $\exists v' \in (b^{-w}, b],\ \exists v'_1, \ldots, v'_k \in D_{x_1} \times \cdots \times D_{x_k} \mid c(v', v'_1, \ldots, v'_k)$
où a^{+w} (resp.a^{-w}) dénote le $(w+1)^{ieme}$ floattant après (resp. avant) a.
Un CSP est 2–B(w)–Consistant ssi tous ses domaines sont 2–B(w)–Consistants.

On peut remarquer que la 2–B(0)–Consistance correspond à la 2–B–Consistance.

Fig. 4 – illustration de la notion de borne "large"

L'algorithme de filtrage par 2B(w)–Consistance s'obtient en modifiant la fonction Narrow pour que la réduction du domaine ne soit effectuée que lorsqu'elle est supérieure à w.

4.2 Problèmes de la 2B(w)–Consistance

Le fait de ne plus réduire le plus possible les intervalles à chaque étape fait perdre la notion de point fixe. En effet, le filtrage par 2–B(w)–Consistance dépend de l'ordre d'évaluation des fonctions de projection.

Exemple 4.2.
Soit $c_1 : x = y, \quad c_2 : x = z, \quad c_3 : x^2 = 4$
$D_x = [-10, 2], D_y = [-2, 2], D_z = [-1.5, 2]$ et $w = 1$.

- Ordre 1 : $< c_1, c_2, c_3 >$
  D'où $Q = \{< c_1, x >, < c_1, y >, < c_2, x >, < c_2, z >, < c_3, x >$
  1. Sélection de $< c_1, x >$, $D_x \leftarrow [-\mathbf{2}, \mathbf{2}]$
  2. Sélection de $< c_1, y >$, D_y ne change pas
     Aucun élément n'est ajouté dans Q.
  3. Sélection de $< c_2, x >$, D_x ne change pas car la réduction est inférieure à w ;
  4. Sélections de $< c_2, z >, < c_3, x >$ ne permettent d'effectuer aucune réduction.
- Ordre 2 : $< c_2, c_1, c_3 >$
  D'où $Q = \{< c_2, x >, < c_2, z >, < c_1, x >, < c_1, y >, < c_3, x >$

1. Sélection de $< c_2, x >$, $D_x \leftarrow [-1.5, 2]$.
   Aucun élément n'est ajouté dans Q.
2. Sélection de $< c_2, z >, < c_1, x >, < c_1, y >$ ne permettent d'effectuer aucune réduction.
3. Sélection de $< c_3, x >$, $D_x \leftarrow [\mathbf{2}, \mathbf{2}]$
   $< c_1, y >$ et $< c_2, z >$ sont ajoutés dans Q.
4. Sélection de $< c_1, y >$, $D_y \leftarrow [\mathbf{2}, \mathbf{2}]$
5. Sélection de $< c_2, z >$, $D_z \leftarrow [\mathbf{2}, \mathbf{2}]$

Remarques :

- Soit $D_x = [a, b]$ le domaine de x dans un CSP P, $[a_1, b_1]$ le domaine de x après filtrage par 2B(w)–Consistance de P, et $[a_2, b_2]$ le domaine de x après filtrage par 2B–Consistance de P. Il est important de comprendre qu'il n'y a pas de rapport direct entre la valeur de w et la distance $|a_1 - a_2|$;
- Soit $D_x = [a, b]$ le domaine de x dans un CSP P, $[a_1, b_1]$ le domaine de x après filtrage par 2B(w_1)–Consistance de P, et $[a_2, b_2]$ le domaine de x après filtrage par 2B(w_2)–Consistance de P. $w_1 < w_2$ n'implique pas $[a_1, b_1] \subset [a_2, b_2]$.
  Ceci est lié à l'enchainement des propagations : un w plus grand peut empêcher la fonction de projection d'une contrainte c_j de réduire le domaine d'une variable x et permettre ainsi à la fonction de projection d'une contrainte c_k d'effectuer une réduction plus significative sur le domaine de cette même variable x.
  Exemple 4.3. Soit $f_1 : y \leftarrow 0.71 \times x \quad f_2 : y \leftarrow 0.6 \times x + z$
  $D_x = [0, 10] \quad Dy = [-10, 20] \quad D_Z = [0, 0.9]$
  Si $w = 2$, f_1 peut réduire D_y à $[0, 7.1]$ et f_2 ne peut plus effectuer aucune réduction.
  Si $w = 3$, f_1 ne peut effectuer aucune réduction et f_2 peut réduire D_y à $[0.8, 6.9]$
- Le mode de décomposition d'un système de contraintes peut avoir une influence sur l'ordre d'évaluation des fonctions de projection et peut de ce fait affecter le résultat d'un filtrage par 2B(w_1)–Consistance ;
- Si les nombres manipulés ont des valeurs très hétérogènes, il est indispensable d'utiliser une valeur relative pour w (e.g., un nombre de flottants) car une valeur absolue risque d'être plus petite que $|f - f^+|$ si f est un grand nombre.

Il est aussi important de noter que ces problèmes ne se posent pas lorsque $w = 1$ car dans ce cas on effectue à chaque fois la réduction maximale

5 3B–Consistance

La 2B–Consistance est uniquement une consistance partielle et comme la consistance d'arc elle est souvent trop faible pour permettre une approximation fine du domaine des variables. De même que des consistances plus fortes, basées sur l'arc consistance (e.g., consistance de chemin [22]), ont été introduites dans les domaines finis, des consistances plus fortes ont été proposées "au dessus" de la 2B–Consistance.

5.1 3B–Consistance[18]

Le principe de la 3B–Consistance est de vérifier si le système de contraintes ne devient pas trivialement inconsistant (i.e., la 2B–Consistance permet de générer des domaines vides) lorsqu'une variable est instanciée par la valeur d'une des bornes de l'intervalle qui lui est associée.
En d'autres termes, la 3B–Consistance cherche à vérifier si une borne peut effectivement être solution.

Définition 5.1 (3B–Consistance). Soit $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP et x une variable de $\mathcal{X}$ de domaine $[a, b]$. Soit :

- $P_{D_x^1 \leftarrow [a,a^+)}$ le CSP derivé de P en substituant D_x dans $\mathcal{D}$ par $D_x^1 = [a, a^+)$;
- $P_{D_x^2 \leftarrow (b^-,b]}$ le CSP derivé de P en substituant D_x dans $\mathcal{D}$ par $D_x^2 = (b^-, b]$.

D_x est 3B–Consistant ssi $\Phi_{2B}(P_{D_x^1}) \neq P_\emptyset$ et $\Phi_{2B}(P_{D_x^2}) \neq P_\emptyset$.
Un CSP est 3–B–Consistant ssi tous ses domaines sont 3–B–Consistants.

Exemple 5.1. Dans l'exemple 2.2, P_1 et P_2 ne sont pas 3–B–Consistants mais P_3 est 3–B–Consistant : $P_3 = \Phi_{3B}(P_1) = \Phi_{3B}(P_2)$.

Il résulte de la definition que tout CSP qui est 3B–Consistant est aussi 2B–Consistant ([18]). La généralisation de la 3–B–Consistance à la k–B–Consistance est immédiate [18, 21].

Définition 5.2 (filtrage par 3B–Consistance). Un filtrage par 3–B–Consistance de P est un CSP P' qui vérifie :

1. $P' \equiv P$,
2. P' est 3–B–Consistant,
3. $P' \preceq P$,
4. $\forall \ P'' = (\mathcal{V}, \mathcal{D}'', \mathcal{C}), \{P'' \equiv P \ \wedge \ (P'' \text{ est 3–B–Consistant}) \ \wedge \ P'' \preceq P \ \wedge \ P' \preceq P''\} \Rightarrow D'' = D'$.

L'algorithme de fermeture par 3–B–Consistance de P est donné par la figure 5.

Cet algorithme repose sur une suite de tentatives de refutation de certaines portions des domaines sans créer de points de choix. Soit $D_x =$

```
function 3B–filtering(P)
% Pré-condition : P = (V, D, C) est un CSP
% Post-condition : le résultat est un CSP P = (V, D, C)  3B consistant
      % S : Vecteur des Δ utilisés pour la réduction des domaines
      % n  :  cardinal de V
1     PF ← false      % PF :  vecteur de booléens qui indique si le point fixe
                         est atteint pour chaque domaine
2     while ¬PF  do
3         for  i = 1 ... n  do
4             if ¬PF_{x_i} then     S_{x_i} ← min(S_{x_i}, size(D_{x_i}))/2
                                    % Valeur initiale des S_{x_i} : maxvalue
5                                   if S_{x_i} ≤ w then      PF_{x_i} ← true
6                                                            S_{x_i} ← w
7         D ← 3B–fixpoint(P, S)
8     return P


function 3B–fixpoint(P, S)
% Pré-condition : P = (V, D, C)  est un CSP
% Post-condition : le résultat est un CSP P = (V, D, C) | P = Φ2(P)
1     PF ← false      % PF : booléen qui indique si le point fixe est atteint
2     while ¬PF  do
3         PF ← true
4         for  i = 1 ... n  do     % n       : cardinal de V
5             P_{borne_inf} = (V, {D_{x_1}, ..., D_{x_{i-1}}, [a, a + S_{x_i}], D_{x_{i+1}}, ..., D_{x_n}}, C)
6             if 2B–filtering(P_{borne_inf}) = P_∅ then
7                 PF ← false
8                 D_{x_i} ← D_{x_i} \ [a, a + S_{x_i}]
9                 D ← 2B–filtering(P)
10            P_{borne_sup} = (V, {D_{x_1}, ..., D_{x_{i-1}}, [b − S_{x_i}, b], D_{x_{i+1}}, ..., D_{x_n}}, C)
11            if 2B–filtering(P_{borne_sup}) = P_∅ then
12                PF ← false
13                D_{x_i} ← D_{x_i} \ [b − S_{x_i}, b]
14                D ← 2B–filtering(P)
15    return D
```

Fig. 5 – Algorithme de filtrage par 3–B–Consistance

$[a, b]$ dans un CSP P, si $\Phi_{3B}(P_{D_x \leftarrow [a, \frac{a+b}{2}]}) = \emptyset$, alors la portion $[a, \frac{a+b}{2})$ de D_x est éliminé et le filtrage se poursuit sur l'intervalle $[\frac{a+b}{2}, b]$; sinon le filtrage se poursuit avec l'intervalle $[a, \frac{a+b}{4}]$. La figure 6 présente les premières étapes d'une réduction d'un intervalle $[a, b]$ à partir de la borne gauche : $\Phi_{3B}(P_{D_x \leftarrow [a,a']}) = \emptyset$, $\Phi_{3B}(P_{D_x \leftarrow [a',c]}) \neq \emptyset$, $\Phi_{3B}(P_{D_x \leftarrow [a',d]}) \neq \emptyset$, $\Phi_{3B}(P_{D_x \leftarrow [a',e]}) = \emptyset$,...

L'algorithme s'arrête lorsque le point fixe est atteint ou lorsque l'intervalle qu'on essaye de refuter devient plus petit qu'une valeur absolue (ou relative) w fixée. Dans ce dernier cas on parle de 3–B(w)–Consistance. Différents réglages de l'algorithme sont possibles en choisisant des w différents pour les critères d'arrêt de l'algorithme de 3–B–Consistance et l'algorithme de 2–B–Consistance que ce dernier utilise : la 3–B(w_1, w_2)–Consistance utilise w_1 comme critère d'arrêt pour la division des domaines et w_2 comme critère d'arrêt pour l'algorithme de 2–B–Consistance utilisé [21].

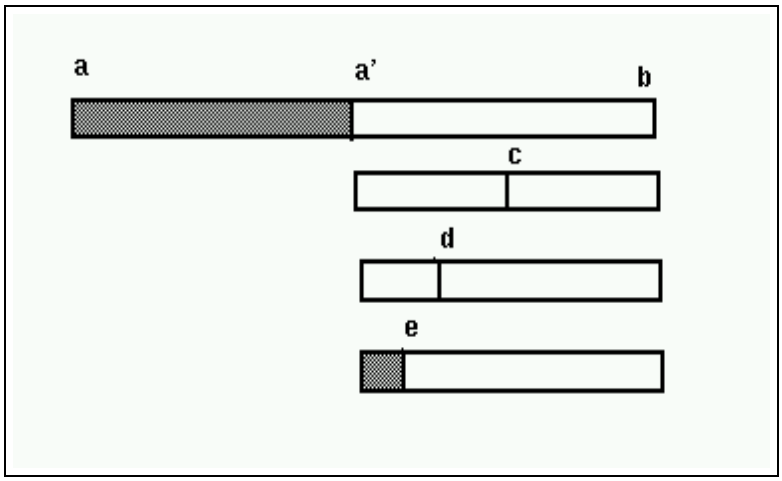

Fig. 6 – Exemple de réduction de la borne gauche

L'exemple ci-dessous montre clairement que la 3–B–Consistance permet d'effectuer des réductions qu'il n'était pas possible d'effectuer localement (i.e., au niveau d'une seule contrainte).

Exemple 5.2. Soit $\mathcal{C} = \{x - y = 0, x + y = 100\}, D_x = D_y = [0, 100]$.
Ce système est 2–B–Consistant mais non 3–B–Consistant : la 3–B–Consistance permet d'approximer avec précision l'unique solution.

Remarques :
Il est facile de voir que la 3–B–Consistance calcule la consistance totale si le système de contraintes n'a que deux variables. De manière générale, on peut montrer que la k–B–Consistance fournit une procédure pour décider de la satisfiabilité d'un système de $k - 1$ variables.

Une question intéressante est celle de la relation entre $\Phi_{2B}(\mathrm{P})$ et $\Phi_{3B}(P_{decomp})$: les exemples 5.3 et 5.4 montrent respectivement que ni la relation $\Phi_{2B}(\mathrm{P}) \preceq \Phi_{3B}(P_{decomp})$, ni la relation $\Phi_{3B}(P_{decomp}) \preceq \Phi_{2B}(P)$ n'est vérifié. Il en résulte qu'aucune relation d'ordre entre $\Phi_{3B}(P_{decomp})$ et $\Phi_{2B}(\mathrm{P})$ ne peut être établie ; même si une seule variable a des occurrences multiples dans les contraintes de P.

Exemple 5.3. Soit P le CSP défini par $\mathcal{C} = \{x_1+x_2 = 10; x_1+x_1-2\times x_2 = 0\}$, $D_{x_1} = D_{x_2} = [0, 10]$. $decomp(x_1 + x_1 - 2 \times x_2 = 0) = \{x_1 + x_3 - 2 \times x_2 = 0, x_3 = x_1\}$. P est 2B–Consistant mais P_{decomp} n'est pas 3B–Consistant : En effet, lorsque x_1 est fixé à 10, $\Phi_{2B}(P_{D_{x_1} \leftarrow [10^-,10]}) = P_\emptyset$ car D_{x_2} est réduit à $\emptyset$. Dans ce cas le lien entre x_1 et x_3 est préservé et la 3B–Consistance reduit D_{x_2} à [5,5].

Exemple 5.4. Soit c la contrainte $x_1 * x_2 - x_1 + x_3 - x_1 + x_1 = 0$; $D_{x_1} = [-4, 3]$, $D_{x_2} = [1, 2]$ et $D_{x_3} = [-1, 5]$ les domaines de ses variables. $decomp(c) = \{x_1*x_2-x_4+x_3-x_5+x_6 = 0, x_1 = x_4 = x_5 = x_6\}$. c n'est pas 2B–Consistant car aucun couple de valeurs de D_{x_1} et de D_{x_2} ne vérifie la relation lorsque $x_3 = 5$.
Toutefois, $decomp(c)$ est 3B–Consistant. La perte du lien entre les deux occurrences de x_1 interdit la réduction du domaine de x_3.

6 La Box–Consistance

La Box–Consistance a été définie pour remédier aux problèmes induits par la décomposition du système de contraintes en contraintes primitives lors du calcul de la 2B–Consistance. La Box–Consistance[1, 31] est une approximation plus grossière de la consistance d'arc que la 2B–Consistance.
Schématiquement, la Box–Consistance peut être définie en remplaçant dans la définition de la 2B–Consitance toutes les variables quantifiées existentiellement (sauf une) par l'intervalle qui correspond à leur domaine. La Box–Consistance génère un système de fonctions univariables qui peut être résolu par des méthodes numériques telle que la méthode de Newton. Contrairement à la 2B–Consistance, la Box–Consistance n'amplifie pas le problème de localité des consistances partielles car elle ne requiert aucune décomposition du système de contraintes. Certains problèmes de dépendance peuvent ainsi être traités efficacement lorsqu'une seule variable a des occurrences multiples.

6.1 Définition et propriétés de la Box–Consistance

Définition 6.1 (Box–Consistance). Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP et $c \in \mathcal{C}$ une contrainte k–aire définie sur les variables $(x_1, \ldots, x_k)$. c est Box–Consistant si, pour tout x_i dans $\{x_1, \ldots, x_k\}$ tel que $D_{x_i} = [a, b]$, les relations ci-dessous sont vérifiées :
1. $C(D_{x_1}, \ldots, D_{x_{i-1}}, [a, a^+), D_{x_{i+1}}, \ldots, D_{x_k})$,
2. $C(D_{x_1}, \ldots, D_{x_{i-1}}, (b^-, b], D_{x_{i+1}}, \ldots, D_{x_k})$.

Le filtrage par Box–Consistance de P est défini de manière similaire au filtrage par 2B–Consistance de P, et est noté $\Phi_{Box}(P)$.
La propriété suivante résulte des définitions :

Proposition 4. $\Phi_{2B}(\mathrm{P}) \preceq \Phi_{Box}(\mathrm{P})$ et $\Phi_{2B}(\mathrm{P}) \equiv \Phi_{Box}(\mathrm{P})$ si aucune variable n'a d'occurrences multiples dans les contraintes de $\mathcal{C}$.

Tout CSP qui est 2B–Consistant est aussi Box–Consistant, alors qu'un CSP peut être Box–Consistant sans être 2B–Consistant (cf. exemple 6.1).

Exemple 6.1. L'exemple 3.4 n'est pas 2B–Consistant pour x_2 mais est Box–Consistant pour x_2 car $([-1,1] \oplus [0,0^+] \ominus [-1,1]) \cap [0,0]$ et $([-1,1] \oplus [1^-,1] \ominus [-1,1]) \cap [0,0]$ ne sont pas vides.

Toutefois, la décomposition du système de contraintes amplifie les limites dues au caractère local de la 2B–Consistance.Il en résulte que la 2B–Consistance sur le système décomposé est plus faible que la Box–Consistance sur le système initial :

Proposition 5. $\Phi_{Box}(\mathrm{P}) \preceq \Phi_{2B}(P_{decomp})$

Preuve : Pour le calcul des consistances locales, CSP P_{decomp} est une relaxation de P. D'où : $\Phi_{Box}(\mathrm{P}) \preceq \Phi_{Box}(P_{decomp})$. Or du fait que P_{decomp} ne contient pas d'occurrences multiples de variables, on a : $\Phi_{Box}(P_{decomp}) \equiv \Phi_{2B}(P_{decomp})$, et donc $\Phi_{Box}(P) \preceq \Phi_{2B}(P_{decomp})\diamond$

Exemple 6.2. Soit c la contrainte $x_1 + x_2 - x_1 - x_1 = 0$, $D_{x_1} = [-1,1]$ et $D_{x_2} = [0.5,1]$ les domaines des variables de c. c n'est pas Box–Consistant car $[-1,-1^+] \oplus [0.5,1] \ominus [-1,-1^+] \ominus [-1,-1^+] \cap [0,0]$ est vide. Par contre, $decomp(c)$ est 2B–Consistant pour D_{x_1} and D_{x_2}.

La Box—Consistance reste toutefois une consistance locale, et on peut montrer [9] que $\Phi_{3B}(P_{decomp}) \preceq \Phi_{Box}(\mathrm{P})$.

Néanmoins, la Box–Consistance peut traiter efficacement certains problèmes de dépendance dans une contrainte c qui ne contient qu'une variable avec des occurrences multiples. Plus précisément, la Box–Consistance peut réduire le domaine D_x si la variable x apparait plusieurs fois dans c et si D_x contient des valeurs inconsistantes (au regard de la contrainte c). Ainsi dans l'exemple 6.2, un filtrage par Box–Consistance réduit le domaine de x_1 car la valeur -1 de D_{x_1} n'a pas de support dans le domaine D_{x_2}.
Toutefois, la Box–Consistance risque d'être inefficace pour traiter les problèmes de dépendance si les valeurs inconsistantes (au regard de la contrainte c) sont dans le domaine d'une variable x_i alors que la variable x_j $(j \neq i)$ apparait plusieurs fois dans c. Ainsi, dans l'exemple 3.4, $(x_1 + x_2 - x_1 = 0$ avec $D_{x_1} = [-1,1], D_{x_2} = [0,1])$ la valeur 1 de D_{x_2} n'a pas de support dans le domaine D_{x_1} mais la Box–Consistance n'arrive pas à le détecter car $[-1,1] \oplus [1^-,1] \ominus [-1,1] \cap [0,0]$ est vrai.

6.2 Calcul de la Box–Consistance

L'algorithme proposé dans [1, 31] pour effectuer un filtrage par Box–Consistance est basé sur l'algorithme générique `IN` (cf. figure 2) utilisé pour la 2B–Consistance [1]. La fonction $narrow(c, \mathcal{D})$ réduit les domaines des variables de c jusqu'à ce que c soit Box–Consistant.
Schématiquement, pour chaque variable x de la contrainte c une fonction univariable sur intervalles est générée en remplaçant toutes les variables sauf x par leur domaine. Le processus consiste alors à trouver le zéro le plus à gauche et la plus à droite dans toutes ces fonctions univariables sur intervalles qui sont de la forme :

$$C(D_{x_1}, .., D_{x_{i-1}}, x, D_{x_{i+1}}, ..., D_{x_k}) = \tilde{0}.$$

La figure 7 décrit la procédure `LNAR` qui calcule le zéro le plus à gauche de F_x pour la variable x de domaine initial I_x. La fonction `LNAR` réduit d'abord le domaine de l'intervalle I_x avec la fonction $NEWTON(F_x, I_x)$ qui correspond à l'extension aux intervalles de la méthode de Newton (cf. figure 8). Il est toutefois possible que la fonction $NEWTON(F_x, I_x)$ ne puisse pas réduire suffisamment le domaine I_x pour le rendre Box–Consistant. C'est pourquoi une étape une division du domaine est effectuée pour s'assurer que la borne gauche de I_x est effectivement un zéro. La fonction `SPLIT` divise l'intervalle I en deux intervalles I_1 et I_2 ; I_1 étant la partie gauche de l'intervalle initial. Ce processus de division permet d'éviter la recherche d'une "safe starting box" pour Newton (cf. [15]). On peut noter que même si F_x n'est pas différentiable, la fonction `LNAR` va trouver le zéro le plus à gauche grâce au processus de division.

```
function LNAR (IN : F_x, I_x, return Interval)
    r ← right(I_x)
    if  0 ∉ F_x(I_x) then return ∅
        else  I_x ← NEWTON(F_x, I_x)
              if 0 ∈ F_x([left(I), left(I)^+]) then return [left(I), r]
                 else  SPLIT(I_x, I_1, I_2)
                       L_1 ← LNAR(F_x, I_1)
                       if L_1 ≠ ∅ then return [left(L_1), r]
                          else return [left(LNAR(F_x, I_2)), r]
                       endif
              endif
    endif
```

Fig. 7 – Function `LNAR`

La Box(w)–Consistance peut être définie de manière similaire à la 2–B(w)–Consistance.

[1] Dans `Numerica`, une heuristique de "domain splitting" est combinée avec cet algorithme de filtrage pour trouver toutes les solutions.

Soit :$f : \mathcal{R} \rightarrow \mathcal{R}$.
D'après le théorème, on a pour toute valeur a entre u et v :

$$f(u) - f(v) = (u - v)f'(a)$$

Si v est un zéro de f on obtient : $v = u - \frac{f(u)}{f'(a)}$.
D'après l'arithmétique des intervalles, si $a \in I$ alors $f(a) \in F(I)$. De plus si F est l'extension naturelle aux intervalles de f, alors $v \in \tilde{u} - \frac{F(\tilde{u})}{F'(I)} = N(F, F', \tilde{u}, I)$. Il en résulte que v est un zéro de f si $v \in N^*(F, F', I)$ avec :
$N^*(F, F', I) = I_n (n \geq 1)$ où :

$I_0 = I$
$I_{i+1} = N(F, F', center(I_i), I) \cap I_i$
$I_n = I_{n-1}$

Fig. 8 – Rappel du principe de l'opérateur itératf de Newton

7 Bound–Consistance

Comme la 2B–Consistance, la Box–Consistance a été généralisée pour limiter les effets de localité. Schématiquement, la Bound–Consistance applique le principe de la 3B–Consistance à la Box–Consistance : elle vérifie si la Box–Consistance peut être appliquée lorsque le domaine d'une variable est réduit à la valeur d'une de ses bornes.

Définition 7.1 (Bound–Consistance). Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP et $c \in \mathcal{C}$ une contrainte d'arité k définie sur les variables $(x_1, \ldots, x_k)$. c est Bound–Consistant si pour tout $x_i \in (x_1, \ldots, x_k)$ avec $D_{x_i} = [a, b]$, les propriétes suivantes sont vérifiées :
1. $\Phi_{Box}(C(D_{x_1}, \ldots, D_{x_{i-1}}, [a, a^+), D_{x_{i+1}}, \ldots, D_{x_k})) \neq P_\emptyset$,
2. $\Phi_{Box}(C(D_{x_1}, \ldots, D_{x_{i-1}}, (b^-, b], D_{x_{i+1}}, \ldots, D_{x_k})) \neq P_\emptyset$.

Comme $\Phi_{Box}(\mathrm{P}) \preceq \Phi_{2B}(P_{decomp})$ on a : $\Phi_{Bound}(P) \preceq \Phi_{3B}(P_{decomp})$.

8 Efficacité de la Box–Consistance et de la 2B–Consistance

L'objectif des solveurs de CSP numériques n'est pas de calculer des propriétés de consistance partielle mais de rechercher les solutions du système de contraintes ; c'est-à-dire de calculer soit les plus petits intervalles qui contiennent, soit des solutions isolées, soit des ensembles de solutions continus.

C'est pourquoi, les systèmes opérationnels (e.g., `Numerica` [31], `PROLOG IV` [28]) combinent des phases de filtrage local avec différentes heuristiques de recherche et des techniques de "domain splitting". Il est difficile de comparer l'efficacité des algorithmes de filtrage en se basant sur les benchmarks de ces différents systèmes car ces derniers diffèrent sur de nombreux points critiques :

- Ils utilisent différentes heuristiques de "domain splitting" ;
- Les implémentations des algorithmes de filtrage sont très différentes : paramétrage de la précision pour arrêter de manière prématurée la propagation (e.g., w), ordonnancement des contraintes, détection de cycles,...
- Ils sont implantés dans des langages de programmation différents : `Prolog`, `C`, ...).

8.1 Coûts de calcul

De ce fait nous allons nous limiter ici à l'examen d'un certain nombre de points clés qui permettent de mieux comprendre les performances de ces systèmes :

1. Coût de l'opérateur de "narrowing" :

   Le coût de l'évaluation de l'opérateur de Newton est supérieur à celui de l'évaluation d'une fonction de projection sur des contraintes primitives. Par exemple, soit la contrainte $\mathcal{C} = \{c^2 = 2\}$ and $D_x = [1, 10]$, le calcul du filtrage par Box–Consistance requiert 6 étapes de réduction avec la méthode de Newton, soit environ 100 opérations sur les intervalles, alors que le filtrage par 2B–Consistance nécessite uniquement le calcul d'une racine carrée et une opération d'intersection sur les intervalles [13].
   De ce fait, la 2B–Consistance est plus efficace que la Box–Consistance sur des problèmes pour lesquels les fonctions de projection permettent de calculer un résultat précis (e.g., exemple du Pentagone)

2. Expansion du système de contraintes :
   La décomposition du système de contraintes initial peut générer un nombre de contraintes primitives très important si les variables ont de nombreuses occurrences multiples. Considérons, par le classique exemple de "Broyden 160" (160 variables initiales, 160 contraintes). La Box–Consistance va générer 160 variables et 1184 fonctions univariables alors que la 2B–Consistance va générer 2368 variables et 6944 fonctions de projection ternaire. Il est clair qu'une telle expansion du système de contraintes est catastrophique au niveau des performances.

3. Précision des calculs :
   Pour une précision donnée des résultats, l'efficacité de la résolution dépend fortement de la précison des calculs intermédiaires. Par exemple, pour "Broyden 160" la recherche d'intervalles solution de taille inférieure ou égale à 10^{-8} est 10 fois plus rapide lorsque le filtrage par Box–Consistance est effectué avec un w de 10^{-1} qu'avec un w de 10^{-8} [13].

8.2 Techniques d'accélération des algorithmes de filtrage

Il existe différentes techniques pour améliorer l'efficacité des algorithmes de filtrage :

- Utilisation de techniques d'analyse numérique (e.g., formes distribuées et méthode de Taylor pour le calcul de la Box–Consistance, méthode d'extrapolation pour le calcul de la Bound–Consistance et des kB–Consistances) ;
- Détection dynamique des cycles en cas de convergences asymptotiques ;
- Mise en oeuvre concurrente de différents algorithmes.

La détection dynamique des cycles [20] permet d'améliorer les performances de l'ensemble des algorithmes de filtrage basés sur `IN`. La complexité en temps de `IN` se situe entre $\Omega(r \times m)$ et $O(r \times m \times a)$ où r est l'arité des contraintes, m le nombre de contraintes, et a le nombre de flottants du plus grand domaine. Les résultats expérimentaux montrent que les temps de calcul de `IN` sont en général très en dessous de la borne supérieure. Toutefois, en cas de convergences asymptotiques (cf figure 9) le temps de calcul peut s'approcher de la borne supérieure et l'algorithme devient alors inutilisable.

$Y = X \quad (a)$
$Y = 1.001 * X \quad (b)$
$Y = 2 * X \quad (c)$
$Z_1 = e^Y \quad (d)$
$Z_2 = e^{Z_1} \quad (e)$
$D_X = [0, 10] \quad D_Y = (-\infty, +\infty) \quad D_{Z_1} = (-\infty, +\infty) \quad D_{Z_2} = (-\infty, +\infty)$

(a) & $D_X = [0, 10] \longrightarrow D_Y = [0, 10]$
(b) & $D_Y = [0, 10] \longrightarrow D_X = [0, 9.99]$
(c) & $D_Y = [0, 10] \longrightarrow D_X = [0, 5]$
(d) & $D_Y = [0, 10] \longrightarrow D_{Z_1} = [1, e^{10}]$
(e) & $D_{Z_1} = [1, e^{10}] \longrightarrow D_{Z_2} = [e, e^{e^{10}}]$
(a) & $D_X = [0, 5] \longrightarrow D_Y = [0, 5]$
(b) & $D_Y = [0, 5] \longrightarrow D_X = [0, 4.99]$
(c) & $D_Y = [0, 5] \longrightarrow D_X = [0, 2.5]$
(d) & $D_Y = [0, 5] \longrightarrow D_{Z_1} = [1, e^5]$
(e) & $D_{Z_1} = [1, e^5] \longrightarrow D_{Z_2} = [e, e^{e^5}]$
etc.

Fig. 9 – Un phénomène de convergence lente

Intuitivement, on voit que ces phénomènes sont cycliques. Dans l'exemple de la figure 9 le cycle est constitué des 5 contraintes :

$$(a, b, c, d, e).$$

Toutefois, la réduction effectuée sur D_X par la contrainte (c) est plus forte que celle effectuée sur D_X par la contrainte (b) ; il est donc inutile d'effectuer

cette dernière. _ Seules (a), (c), (d) et (e) sont pertinentes et une première simplification consiste à réduire le cycle à (a, c, d, e).

Mais les contraintes (d) et (e) ne permettent que de réduire les domaines de Z_1 et Z_2. Il serait donc plus judicieux de différer leur application jusqu'à la fin du cycle. Le report des contraintes (d) et (e) permet de réduire le cycle à (a, c). Le nombre d'opérations effectués par `IN` serait donc réduit de $O((a, b, c, d, e)^k$ à $O(a, c)^k$ où k est le nombre d'itérations nécessaires pour atteindre le point fixe.

La présence de cycles implique l'existence d'une série $u_k = f(u_{k-1})$ qui converge vers un point fixe u tel que $u = f(u)$. L'équation $u = f(u)$ ne peut malheureusement pas être simplifiée de manière symbolique dans le cas de contraintes quelconques.

Il est toutefois possible de simplifier dynamiquement l'évaluation des termes de la série $u_k = f(u_{k-1})$ pour accélérer la convergence vers le point fixe. Deux types de simplification sont possibles [20] :

- Suppression des fonctions de réduction non pertinentes (e.g., b dans l'exemple 9) ;
- Report des fonctions de réduction qui correspondent à des radicelles du cycle (e.g., d, e dans l'exemple 9)XS.

9 Utilisation des techniques de propagation d'intervalle sur les domaines finis et sur les booléens

Les techniques de propagation d'intervalles on été utilisées sur d'autres domaines que les domaines continus. Nous présentons ici sommairement leurs possibilités d'application pour les domaines finis et les booléens.

9.1 La consistance d'intervalles sur les domaines finis

L'idée consiste ici à calculer rapidement la "boite" qui contient l'ensemble des valeurs arc-consistantes du domaine.

Définition 9.1 (Intervalle–Consistance). Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP et $c \in \mathcal{C}$ une contrainte k-aire sur les variables $(x_1, \ldots, x_k)$. Soit D un domaine fini, et $D^* = \{min(D), .., max(D)\}$. La contrainte c est Intervalle–Consistante ssi : $\forall x_i \in \mathcal{X}, \forall v_i \in \{min(D_i), max(D_i)\}$ $\exists v_1 \in D^*_{x_1}, \ldots, \exists v_{i-1} \in D^*_{x_i-1}, \exists v_{i+1} \in D^*_{x_i+1}, \ldots, \exists v_k \in D^*_{x_k}$ tel que $c(v_1, \ldots, v_{i-1}, x_i, v_{i+1} \ldots, v_k)$ est vérifiée.
Un CSP est Intervalle–Consistant ssi toutes ses contraintes sont Intervalle–Consistantes.

Ce traitement des contraintes sur les domaines finis est bien adapté aux problèmes dits "mixtes" où certaines variables sont définies sur les entiers et d'autres sur les réels.

9.2 Filtrage de système de contraintes booléens

On peut considérer les variables booléennes comme des entiers qui prennent leurs valeurs dans l'intervalle $[0, 1]$. Les relations booléennes sont définies de la manière suivante :

- $and = min = \{(x, y, z) \in \mathcal{R}^3 : min(x, y) = z\}$
- $or = max = \{(x, y, z) \in \mathcal{R}^3 : max(x, y) = z\}$
- $not = \{(x, y) \in \mathcal{R}^2 : y = 1 - x\}$

Le calcul de min et max n'est pas trivial : il faut considérer 96 cas pour ces 3 fonctions car pour chaque couple d'intervalles on peut avoir soit deux intervalles disjoints, soit un recouvrement total entre deux intervalles, soit un recouvrement partiel entre deux intervalles.

Toutefois, on peut exprimer min et max de la manière suivante qui est plus simple à calculer :

- $min = (\{(x, y, z) \in \mathcal{R}^3, x \leq y\} \cap \{(x, y, z) \in \mathcal{R}^3, z = x\}) \cup$
  $(\{(x, y, z) \in \mathcal{R}^3, x > y\} \cap \{(x, y, z) \in \mathcal{R}^3, z = y\})$
- $max = (\{(x, y, z) \in \mathcal{R}^3, x \geq y\} \cap \{(x, y, z) \in \mathcal{R}^3, z = x\}) \cup$
  $(\{(x, y, z) \in \mathcal{R}^3, x < y\} \cap \{(x, y, z) \in \mathcal{R}^3, z = y\})$

Les opérations de base de réduction sur les booléens sont :

- $x \vee 1 = y \Rightarrow y = 1$
- $x \vee 0 = 1 \Rightarrow x = 1$
- $x \wedge y = 1 \Rightarrow x = 1, y = 1$
- $x \vee y = 0 \Rightarrow x = 0, y = 0$

De plus on peut utiliser le fait que les booléens sont représentés par des entiers pour exprimer facilement des contraintes de cardinalité. Soit $(x_1, .., x_n)$ une séquence de booléens. La contrainte $a \leq x_1 + ... + x_n \leq b$ exprime le fait qu'au moins a et au plus b variables x_i doivent être vraies.

D'autres contraintes redondantes utiles sont :

- $a \Rightarrow b : a \leq b$,
- `if a then b else c` : $(2 - a - b) \times (1 - c + a) = 0$;
- a XOR $b : a \neq b$.

Ce traitement numérique des contraintes est particulièrement bien adapté aux problèmes mixtes qui mélangent variables booléennes et variables entières ou réelles.

Exemple 9.1. Soit un problème d'ordonnancement défini par un ensemble de tâches $T = \{t_1, .., t_n\}$. A chaque tâche t_i est associé un couple $< s_i, d_i >$ où s_i correspond au début de la tâche et d_i à la durée. Pour exprimer que deux tâches t_i et t_j partagent la même ressource, il suffit d'écrire :

$$(s_i + d_i \leq s_j) \vee (s_j + d_j \leq s_i) = 1$$

Références

[1] F. Benhamou, D. Mc Allester, and P. Van Hentenryck. CLP(Intervals) Revisited. in Proc. Logic Programming : Proceedings of the 1994 International Symposium,MIT Press, (1994).

[2] F. Benhamou and W. Older. Applying interval arithmetic to real, integer and boolean constraints. Journal of Logic Programming, (1997).

[3] F. Benhamou. Interval Constraint Logic Programming. In Andreas Podelski, editor, Constraint Programming : Basics and Trends, LNCS 910. Springer-Verlag, 1995. (Châtillon-sur-Seine Spring School, France, May 1994).

[4] C. Bliek. Computer Methods for Design Automation. PhD thesis, Massachusetts Institute of Technology, 1992.

[5] R.P. Brent. A FORTRAN multiple-precision arithmetic package ACM Trans. on Math. Software, 4, no 1, 57-70, 1978.

[6] Hélène Collavizza, François Delobel, Michel Rueher. A Note on Partial Consistencies over Continuous Domains. RR I3S n°97-25, Université de Nice, Nov. 1997.

[7] C. K. Chiu and J. H. M. Lee. Towards practical interval constraint solving in logic programming. In ILPS'94 : Proceedings 11th International Logic Programming Symposium, Ithaca, November 1994.

[8] J.C. Cleary. Logical arithmetic. Future Computing Systems, 2(2) :125–149, 1987.

[9] H.Collavizza, F.Delobel, M. Rueher. A Note on Partial Consistencies over Continuous Domains Solving Techniques. Proc. CP98 (Fourth International Conference on Principles and Practice of Constraint Programming), Pisa, Italy, October 26-30, pp. 147–161, 1998.

[10] E. Davis. Constraint propagation with interval labels. Artificial Intelligence, 32(3) :281–331, July 1987.

[11] B. Faltings, 'Arc consistency for continuous variables,'Artificial Intelligence, 65(2), 1994.

[12] E. C. Freuder. Synthesizing constraint expressions. Communications of the ACM, 21 :958–966, November 1978.

[13] L. Granvilliers. On the combination of Box–Consistency and Hull-consistency. Workshop ECAI on non binary-constraints, Brighton, United Kingdom,1998

[14] E. Hansen. Global optimization using interval analysis. Marcel Dekker, NY, 1992.

[15] Hoon Hong and Volker Stahl. Safe starting regions by fixed points and tightening. Computing, 53 :323–335, 1994.

[16] E. Hyvőnen. Constraint reasoning based on interval arithmetic : the tolerance propagation approach. Artificial Intelligence, vol. 58, pp. 71–112, 1992.

[17] J. H. M. Lee and M. H. van Emden. Interval computation as deduction in CHIP. Journal of Logic Programming, 16 :3–4, pp.255–276, 1993.

[18] O. Lhomme. Consistency techniques for numeric CSPs. in Proc. IJCAI93, Chambery, (France), pp. 232–238, (August 1993).

[19] O. Lhomme. Contribution à la résolution de contraintes sur les réels par propagation d'intervalles. Thèse de doctorat, University of Nice Sophia Antipolis - CNRS, Route des Colles, B.P. 145, 06903 Sophia Antipolis Cedex, France,

[20] O. Lhomme, A. Gotlieb and M. Rueher. Dynamic optimization of Interval Narrowing Algorithms Journal of Logic Programming (Elsevier Science Inc). 37(1-3) : 165-183 (1998).

[21] O. Lhomme and M. Rueher. Application des techniques CSP au raisonnement sur les intervalles. RIA (Dunod),vol. 11 :3, pp. 283–312, 1997.

[22] A. Mackworth. Consistency in networks of relations. Artificial Intelligence, vol. 8, no. 1, pp. 99–118, 1977.

[23] U. Montanari. Networks of constraints : Fundamental properties and applications to picture processing. Information Science, 7(2) :95–132, 1974.

[24] R. Moore, Interval Analysis. Prentice Hall, 1966.

[25] A. Neumaier, interval methods for systems of equations Cambridge University Press, 1990.

[26] W.J. Older and A. Velino. Extending prolog with constraint arithmetic on real intervals. In Proc. of IEEE Canadian conference on Electrical and Computer Engineering. IEEE Computer Society Press, 1990.

[27] W. Older and A. Vellino. Constraint arithmetic on real intervals. in Constraint Logic Programming : Selected Research, eds., Frédéric Benhamou and Alain Colmerauer. MIT Press, (1993).

[28] Prologia PrologIV Constaint inside. Parc technologique de Luminy - Case 919 13288 Marseille cedex 09 (France), 1996.

[29] M. Rueher, C. Solnon. Concurrent Cooperating Solvers within the Reals. Reliable Computing. Kluwer Academic Publishers, Vol.3 :3, pp. 325-333, 1997.

[30] E. Tsang. Foundations of Constraint Satisfaction. Academic Press, 1993.

[31] P. Van Hentenryck, Y. Deville, and L. Michel. Numerica. A modelling language for global optimization. MIT Press, 1997.

[32] D.L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Tech. rept. ai-tr-271, IT, Cambridge, MA, 1972.