Stratégies de résolution anytime de problèmes de satisfaction de contraintes numériques

Thomas Richard de Latour[1]*, **Raphaël Chenouard**[2], **Laurent Granvilliers**[1]
Laboratoire des Sciences du Numérique de Nantes, UMR CNRS 6004
Université de Nantes[1], France
Centrale Nantes[2], France
thomas.richard-de-latour@ls2n.fr raphael.chenouard@ec-nantes.fr
laurent.granvilliers@univ-nantes.fr

Résumé

Un algorithme Branch-and-Prune calcule un pavage de l'ensemble des solutions d'un problème de satisfaction de contraintes numériques à une précision donnée. Un algorithme anytime stoppe le calcul prématurément, par exemple en limitant le temps de calcul. Cet article revisite les algorithmes Branch-and-Prune Anytime et introduit deux nouvelles stratégies de recherche ainsi que trois indicateurs de qualité pour les comparer. Une étude de complexité et une analyse expérimentale montrent que ces nouvelles stratégies sont pertinentes.

Abstract

A Branch-and-Prune algorithm computes a paving of the solution set of a numerical constraint satisfaction problem to a given precision. An anytime algorithm prematurely stops the computation to a given criterion (for example a maximum running time). This article revises anytime Branch-and-Prune algorithms and introduces two new search strategies and three quality indicators to compare them. The evaluation of their complexity and an experimental analysis enabled to prove the efficiency of those new strategies.

1 Introduction

Un problème de satisfaction de contraintes numériques (NCSP) est défini par un ensemble de contraintes (équations et inégalités) non linéaires sur des variables réelles dont les domaines sont des intervalles. C'est un cadre utilisé pour modéliser des applications dans de nombreux domaines comme en robotique [5], en automatique [6] et en conception de systèmes [2]. Résoudre un NCSP peut prendre plusieurs sens : prouver qu'il n'existe pas de solutions, trouver une solution, toutes les solutions, la meilleure selon une fonction objectif ou un sous-ensemble de solutions diverses. C'est ce dernier sens que nous privilégions ici. Nous visons des applications en conception de systèmes où un système est souvent modélisé par un NCSP sous-contraint admettant un nombre de solutions infini. Le but est de trouver des architectures différentes et représentatives dans un tel espace de solutions pour aider au choix des bons concepts.

Un algorithme de type Branch-and-Prune (BP) sur les intervalles a pour but de calculer des boîtes (produits cartésiens d'intervalles) encadrant les solutions d'un NCSP à une précision $\epsilon > 0$ fixée [16]. Un arbre de recherche est généré en alternant des phases de réduction des domaines au moyen de techniques de cohérence locale sur les intervalles [8, 1, 15] et des phases de découpage des domaines. Une exploration complète permet de calculer une couverture de l'espace des solutions à la précision voulue mais ce n'est pas réalisable en un temps raisonnable si l'espace des solutions est infini et si la précision ϵ est faible.

Notre travail s'intéresse aux algorithmes Branch-and-Prune anytime (ABP). En fixant un critère d'arrêt comme un temps de calcul limite ou un nombre de solutions maximum, un tel algorithme réalise une exploration partielle de l'espace avec le but d'obtenir des solutions diverses, représentatives, les plus éloignées possibles les unes des autres. Ce type d'algorithme est notamment utilisé pour le calcul de points bien répartis sur un front de Pareto en optimisation multi-objectif [7]. Dans ce but, les parcours classiques de

*Papier doctorant : Thomas Richard de Latour[1] est auteur principal.

l'arbre de recherche ne fonctionnent pas : un parcours en profondeur (DFS, Depth-First Search) convergerait rapidement vers des solutions proches ; un parcours en largeur (BFS, Breadth-First Search) serait très lent pour atteindre des boîtes de précision ϵ. Il existe des stratégies de parcours hybrides pour contrer ces problèmes. Par exemple, la stratégie présentée dans [11] alterne des étapes de recherche en profondeur et des étapes de recherche en largeur pour diversifier (toutefois pour des problèmes discrets). Dans le même esprit, la stratégie DMDFS [3] a pour but de maximiser la distance entre les solutions calculées. Elle alterne des phases de recherche en profondeur pour trouver une solution et des phases de sélection de la prochaine boîte à explorer la plus éloignée des solutions déjà calculées. Dans le cadre de l'optimisation, la stratégie Best-First [12] consiste à sélectionner le prochain nœud de l'arbre selon un critère donné.

Dans ce travail, nous proposons un cadre pour décrire les stratégies ABP ainsi que deux nouvelles stratégies. Elles sont comparées à l'heuristique DMDFS [3] sur le plan de la complexité algorithmique. Une analyse expérimentale est également conduite au moyen d'un prototype développé au sein de la librairie IBEX. Pour qualifier les résultats obtenus, différents indicateurs de qualité sont définis en s'inspirant de l'existant en optimisation multi-objectif [14, 9].

Après un bref rappel des notions de NCSP et de calcul par intervalles dans la section 2, nous présentons les algorithmes ABP et les différentes stratégies de recherche existantes dans la section 3. La section 4 introduit les nouvelles stratégies. Dans la section 5 nous présentons le contenu des expérimentations et les indicateurs de qualité pour qualifier les résultats.

2 Notions Préliminaires

2.1 Arithmétique des intervalles

On note $[x]$ un *intervalle* fermé de $\mathbb{R}$ défini par une borne inférieure $\underline{x}$ et une borne supérieure $\overline{x}$ telles que $[x] = \{y \in \mathbb{R} : \underline{x} \leq y \leq \overline{x}\}$. La *largeur* de $[x]$ est définie par le nombre réel $w([x]) = \overline{x} - \underline{x}$. L'ensemble des intervalles est noté $\mathbb{I}$.

Une *boîte* $[\mathbf{x}]$ de dimension n est un produit cartésien d'intervalles $[\mathbf{x}] = [x_1] \times [x_2] \times \cdots \times [x_n] \subseteq \mathbb{R}^n$. La *largeur* de $[\mathbf{x}]$ est le maximum des largeurs de ses composants. Le *volume* de $[\mathbf{x}]$ est le produit des largeurs de ses composants. Étant donné un nombre réel $\epsilon > 0$, on dit que $[\mathbf{x}]$ est une *ϵ-boîte* si $w([\mathbf{x}]) \leq \epsilon$.

L'*enveloppe* d'un ensemble $S \subseteq \mathbb{R}$ est le plus petit intervalle contenant S noté $\Box S$. L'*enveloppe* d'un ensemble $S \subseteq \mathbb{R}^n$ est la plus petite boîte (en volume) contenant S notée $\Box S$. Par extension, l'enveloppe d'un ensemble de boîtes correspond à l'enveloppe de l'union de ces boîtes. Un *pavage* d'un ensemble $S \subseteq \mathbb{R}^n$ est un ensemble de boîtes de $\mathbb{I}^n$ dont l'union contient S.

La distance de Hausdorff entre deux intervalles est définie par $d_H([x],[y]) = max\{|\overline{x} - \overline{y}|, |\underline{x} - \underline{y}|\}$. La distance de Hausdorff entre deux boîtes $[\mathbf{x}], [\mathbf{y}] \subseteq \mathbb{R}^n$ est définie par

$$d_H([\mathbf{x}],[\mathbf{y}]) = \max\{d_H([x_i],[y_i]) : 1 \leq i \leq n\}.$$

Les opérations arithmétiques réalisent des calculs d'enveloppes. Ainsi, une opération binaire $\diamond$ est étendue aux intervalles de manière à vérifier la propriété $[x] \diamond [y] = \Box\{x \diamond y : x \in [x], y \in [y]\}$ et on a par exemple $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$.

Les fonctions réelles sont également étendues aux intervalles. Étant données deux fonctions $f : \mathbb{R}^n \to \mathbb{R}$ et $[f] : \mathbb{I}^n \to \mathbb{I}$, on dit que $[f]$ est une *extension* de f si pour toute boîte $[\mathbf{x}]$ de dimension n et tout point $x \in [\mathbf{x}]$ on a $f(x) \in [f]([\mathbf{x}])$. L'extension dite *naturelle* consiste à évaluer une expression de f au moyen des opérations intervalles. Par exemple, considérons la fonction $f(x_1, x_2) = x_1 \times (x_2 - 1) + 2$ et $[\mathbf{x}] = [1,3] \times [0,2]$. On calcule

$$[f]([\mathbf{x}]) = [1,3] \times ([0,2] - 1) + 2 = [-1,5]$$

et l'intervalle ainsi obtenu a la propriété d'être un sur-ensemble de l'ensemble image de $[\mathbf{x}]$ par f.

2.2 CSP numérique

Definition 2.1 (NCSP)**.** *Un problème de satisfaction de contraintes numérique ou NCSP est un triplet $\mathcal{P} = \langle \mathcal{X}, [\mathbf{x}^{init}], \mathcal{C} \rangle$ défini par :*

- *un ensemble de variables $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$;*
- *une boîte $[\mathbf{x}^{init}]$ de dimension n telle que $x_j \in [x_j^{init}]$ pour tout j ;*
- *un ensemble de contraintes $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ où chaque contrainte est définie comme une équation ou une inéquation sur $\mathcal{X}$.*

L'ensemble des solutions de $\mathcal{P}$ est défini par $\Sigma_{\mathcal{P}} = \{x \in [\mathbf{x}^{init}] : c_1(x)$ et $c_2(x)$ et $\ldots$ et $c_p(x)\}$.

Résoudre un NCSP $\mathcal{P}$ pour une précision donnée $\epsilon > 0$ consiste à calculer un pavage de $\Sigma_{\mathcal{P}}$ composé de ϵ-boîtes. Ce pavage peut être obtenu par des algorithmes BP alternant des étapes de propagation pour contracter les domaines et des étapes de recherche pour séparer les problèmes en sous-problèmes. Le paragraphe suivant introduit les méthodes de filtrage sur les intervalles. La section 3 concerne les stratégies de recherche.

2.3 Contracteur et propagation

Les contraintes d'un NCSP permettent de filtrer les domaines des variables de manière à supprimer des

valeurs incohérentes. Dans ce but, on définit les notions de *contracteur* et de *propagation*.

Definition 2.2 (Contracteur)**.** *Un contracteur associé à une contrainte c est un opérateur $\gamma_c : \mathbb{I}^n \to \mathbb{I}^n$ qui vérifie les propriétés suivantes pour tout $[\mathbf{x}] \in \mathbb{I}^n$:*

1. $\gamma_c([\mathbf{x}]) \subseteq [\mathbf{x}]$.
2. $\forall x \in [\mathbf{x}] \setminus \gamma_c([\mathbf{x}])$ *: non* $c(x)$.

Il existe différentes techniques de contraction [8, 1, 15]. En particulier, l'algorithme HC4 repose sur un mécanisme d'inversion des contraintes combiné à des évaluations sur les intervalles. Considérons par exemple l'équation $f(x_1, x_2) = x_1 \times (x_2 - 1) + 2 = 0$ et $[\mathbf{x}] = [1, 3] \times [0, 2]$. Il vient $x_2 = 1 - 2/x_1$ et on réduit le domaine de x_2 de la manière suivante :

$$[x_2] \leftarrow [0, 2] \cap \left(1 - \frac{2}{[1, 3]}\right) = \left[0, \frac{1}{3}\right].$$

En associant un contracteur à chaque contrainte d'un NCSP, on peut appliquer ces contracteurs pour réduire les domaines jusqu'à un état stable. Ainsi, les réductions de domaines obtenues avec une contrainte sont propagées aux autres. La stratégie de propagation est en général de type AC3 [10].

3 Travaux connexes

3.1 Branch-and-Prune

L'algorithme 1 calcule un pavage de l'ensemble des solutions d'un NCSP à une précision donnée ϵ. La liste L est composée de sous-boîtes de la boîte initiale $[\mathbf{x}^{init}]$ correspondant aux feuilles de l'arbre de recherche à traiter. Une boîte $[\mathbf{x}]$ est extraite de L à chaque itération. Cette boîte est contractée (filtrée) puis, si elle n'est pas vide, découpée si sa largeur est supérieure à ϵ ou insérée dans le pavage en sortie sinon.

3.2 Algorithme Anytime

Un algorithme BP Anytime a pour but de résoudre partiellement un NCSP $\mathcal{P}$ en fixant généralement un seuil sur un critère comme le temps de calcul, le nombre de solutions, le nombre d'itérations, etc. Le résultat est donc un pavage d'un sous-ensemble de l'ensemble des solutions $\Sigma_{\mathcal{P}}$, c'est-à-dire un *sous-pavage* de $\Sigma_{\mathcal{P}}$.

Pour évaluer les algorithmes BP, on mesure par exemple le temps de calcul ou le nombre de boîtes générées. Pour les algorithmes Anytime, on peut s'inspirer des travaux en optimisation multiobjectif [14, 9]. Nous retenons les trois qualités suivantes :

— l'uniformité de la distribution des boîtes du sous-pavage calculé ;

Algorithme 1 Algorithme Branch-and-Prune

Entrées : NCSP $\mathcal{P} = \langle \mathcal{X}, [\mathbf{x}^{init}], \mathcal{C} \rangle$, précision $\epsilon > 0$
Sortie : pavage Σ de l'ensemble $\Sigma_{\mathcal{P}}$

$L \leftarrow \{[\mathbf{x}^{init}]\}$;
$\Sigma \leftarrow \emptyset$;
tant que $(L \neq \emptyset)$ **faire**
  $[\mathbf{x}] \leftarrow$ **ExtrairePremier**(L) ;
  Contracter$_{\mathcal{C}}([\mathbf{x}])$;
  si $[\mathbf{x}]$ n'est pas vide **alors**
    si $w([\mathbf{x}]) \leq \epsilon$ **alors**
      $\Sigma \leftarrow \Sigma \cup \{[\mathbf{x}]\}$;
    sinon
      $\{[\mathbf{x}_1], [\mathbf{x}_2]\} \leftarrow$ **Diviser**$([\mathbf{x}])$;
      $L \leftarrow L \cup \{[\mathbf{x}_1], [\mathbf{x}_2]\}$;
    finsi
  finsi
fin tant que

— le taux de couverture de l'ensemble des solutions du NCSP ;
— la cardinalité du sous-pavage calculé.

La figure 1 illustre le résultat obtenu avec un ABP pour la résolution du NCSP

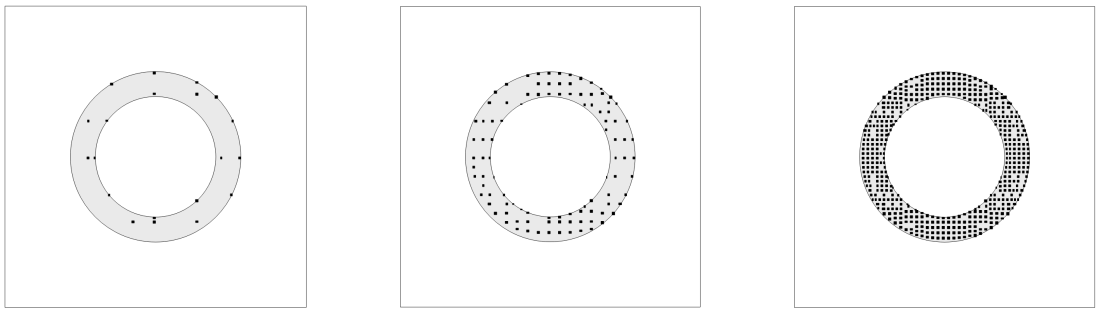
$$\mathcal{P}_1 \begin{cases} x^2 + y^2 \leq 4 \\ x^2 + y^2 \geq 2 \\ -5 \leq x, y \leq 5 \end{cases} \quad (1)$$

selon deux stratégies différentes. Les figures 1d, 1e, 1f sont obtenues par la stratégie Depth-First Search (DFS) de parcours en profondeur et les figures 1a, 1b, 1c par l'algorithme Anytime IDFS, présenté dans la section suivante.

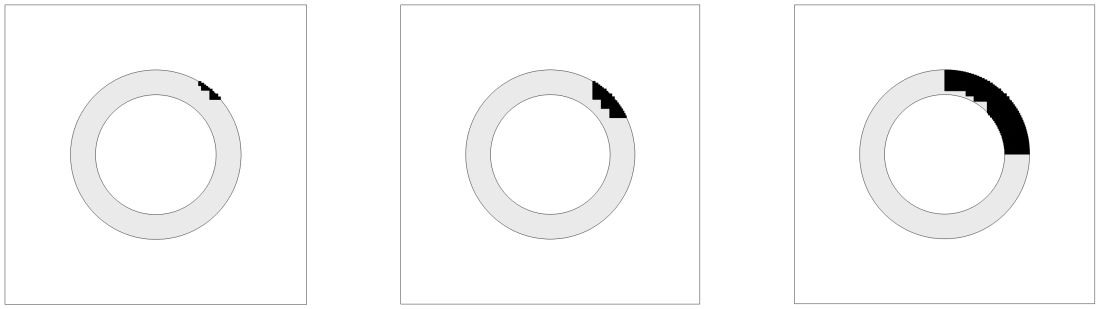
3.3 Stratégies de recherche

Dans l'algorithme 1, la stratégie de recherche est paramétrée par les algorithmes **ExtrairePremier** sélectionnant la prochaine boîte à traiter et **Diviser** séparant la boîte courante en sous-boîtes. Notre étude se concentre sur les stratégies pour sélectionner la prochaine boîte à traiter dans un algorithme BPA et on utilise par exemple une technique de bissection du plus grand domaine pour diviser les boîtes.

Best-First Search La stratégie de recherche *Best-First* (BestFS) a été introduite dans [4] pour diriger l'exploration de graphes au moyen d'une fonction d'évaluation ρ. BestFS est surtout utilisé en optimisation pour accélérer la convergence vers les solutions optimales. Dans le cas des NCSP, il s'agit de maximiser la qualité du sous-pavage calculé au fur et à mesure de



(a) 20 ϵ-boîtes (b) 100 ϵ-boîtes (c) 500 ϵ-boîtes



(d) 20 ϵ-boîtes (e) 100 ϵ-boîtes (f) 500 ϵ-boîtes

FIGURE 1 – Approximations de la résolution du NCSP $\mathcal{P}_1$ (1). Les figures a, b et c illustrent une stratégie de recherche Anytime. Les figures d, e, et f ont été obtenue avec la stratégie DFS

l'exploration. L'algorithme 2 montre l'implémentation de cette stratégie dans un algorithme ABP. Les boîtes non traitées de la liste L sont évaluées par $\rho : \mathbb{I}^n \to \mathbb{R}$ et cette liste est ordonnée sur ce critère.

Les heuristiques BestFS trouvent un intérêt particulier dans l'exploration Anytime d'espace de recherche. La fonction d'évaluation ρ doit tendre vers l'amélioration de la qualité de l'ensemble Σ à chaque nouvelle solution calculée.

En pratique ces stratégies donnent de bonnes approximations mais manquent d'efficacité pour trouver les ϵ-boîtes. Une manière de contourner ce problème est d'implémenter une version hybride DFS/BestFS de ces heuristiques [13]. L'algorithme 3 en découle. L'idée est de trier la liste L seulement après la découverte d'une nouvelle ϵ-boîte solution.

Depth and Most-Distant-First Search Notre travail se positionne dans le sillage de la stratégie *Most-Distant-First Search* (MDFS) [3] associant comportement Anytime et exploration d'espace de recherche. La prochaine boîte sélectionnée par MDFS est celle qui maximise la fonction d'évaluation ρ_M définie par

$$\rho_M([\mathbf{x}], \Sigma) = min\{d_H([\mathbf{x}], \sigma) : \sigma \in \Sigma\} \qquad (2)$$

où Σ est le sous-pavage courant de l'ensemble des solutions et $[\mathbf{x}] \in L$.

Algorithme 2 Algorithme Best-First Search Anytime Branch & Prune

Entrées : NCSP $\langle \mathcal{X}, [\mathbf{x}^{init}], \mathcal{C} \rangle$, Précision ϵ, Critère d'arrêt ϕ

Sortie : sous-pavage Σ de l'ensemble $\Sigma_{\mathcal{P}}$

$L \leftarrow \{[\mathbf{x}^{init}]\}$;
$\Sigma \leftarrow \emptyset$;
tant que ($L \neq \emptyset$ et $\neg\phi$) **faire**
  $[\mathbf{x}] \leftarrow$ **ExtrairePremier**(L) ;
  Contracter$_C([\mathbf{x}])$;
  si $[\mathbf{x}]$ n'est pas vide **alors**
    si w$([\mathbf{x}]) \leq \epsilon$ **alors**
      $\Sigma \leftarrow \Sigma \cup [\mathbf{x}]$;
    sinon
      $\{[\mathbf{x}_1], [\mathbf{x}_2]\} \leftarrow$ **Diviser**$([\mathbf{x}])$;
      $L \leftarrow L \cup \{[\mathbf{x}_1], [\mathbf{x}_2]\}$;
      Evaluer$(\rho([\mathbf{x}_1]))$;
      Evaluer$(\rho([\mathbf{x}_2]))$;
      Trier(L, ρ) ;
    finsi
  finsi
fin tant que
Sortie : Σ
Fin

Algorithme 3 Algorithme hybride BestFS/DFS statique

```
...
  si w([x]) ≤ ϵ alors
    Σ ← Σ ∪ {[x]} ;
    Trier(L, ρ) ;
  sinon
    {[x1], [x2]} ← Diviser([x]) ;
    L ← L ∪ {[x1], [x2]} ;
    Evaluer(ρ([x1])) ;
    Evaluer(ρ([x2])) ;
  finsi
...
```

MDFS peut être assimilée à une stratégie BestFS ayant pour fonction d'évaluation ρ_M. Afin de limiter le nombre d'évaluations et obtenir rapidement les premières solutions, une version hybride DFS/MDFS est proposée dans [3]. Cette version appelée *Depth and Most-Distant-First Search*(DMDFS) est implémentée par l'algorithme 4 qui consiste à évaluer les boîtes non explorées avant l'étape de tri.

Complexité des stratégies Deux catégories de stratégies BestFS peuvent être identifiées : le cas *statique* où la valeur associée à une boîte par la fonction ρ est constante pour toute l'exploration ; le cas *dynamique* qui demande de réévaluer les critères de l'ensemble des boîtes non explorées par ρ à chaque itération (ligne 12 dans l'algorithme 2). Les stratégies dynamiques permettent de prendre en compte des informations qui évoluent au fur et à mesure de l'exploration et ainsi adapter l'exploration à l'aide de ρ, à l'image de DMDFS qui réordonne les boîtes de L selon ρ_M après chaque nouvelle solution trouvée. Ainsi, si ρ est à coût constant, l'étape d'évaluation d'une heuristique dynamique a un coût linéaire en fonction du nombre de boîtes non explorées contre un coût constant pour une heuristique statique.
Pour DMDFS la complexité de ρ_M dépend du nombre de variables n, du nombre de solutions déjà calculées m et de la taille des liste des boîtes non-explorées p. Pour cette étape on distingue :

1. l'évaluation des boîtes issues de la bissection précédente, pour lesquelles on calcule entièrement la valeur de ρ_M. Complexité en $O(nm)$;
2. la mise à jour de l'évaluation des boîtes non-explorées, où seule la distance à la dernière solution est calculée. Complexité en $O(np)$.

La complexité de l'étape d'évaluation de DMDFS est donc en $O(nm + np)$.

Algorithme 4 Algorithme hybride BestFS/DFS dynamique

```
...
  si w([x]) ≤ ϵ alors
    Σ ← Σ ∪ {[x]} ;
    Evaluer(ρ([x])), ∀ [x] ∈ L ;
    Trier(L, ρ) ;
  sinon
    {[x1], [x2]} ← Diviser([x]) ;
    L ← L ∪ {[x1], [x2]} ;
  finsi
...
```

4 Stratégies de recherche pour algorithmes BPA

Cette section introduit de nouvelles stratégies adaptées à l'exploration Anytime de CSP numériques. Elles s'appuient sur l'algorithme hydride BestFS/DFS dont les fonctions d'évaluations maximisent la qualité des approximations.

4.1 Interleaved Depth-First Search

Cette stratégie s'inspire des travaux de [11], une heuristique d'exploration d'arbres de recherche appelée *Interleaved Depth-First Search* (IDFS). IDFS propose une exploration hybride entre un parcours en profondeur et un parcours en largeur de l'arbre. Le parcours en profondeur permet d'atteindre rapidement les premières solutions, tandis que le parcours en largeur est réalisé par ordre croissant de la profondeur des nœuds dans l'arbre. Concrètement IDFS est assimilable à une stratégie hybride DFS/BestFS dont la fonction d'évaluation est :

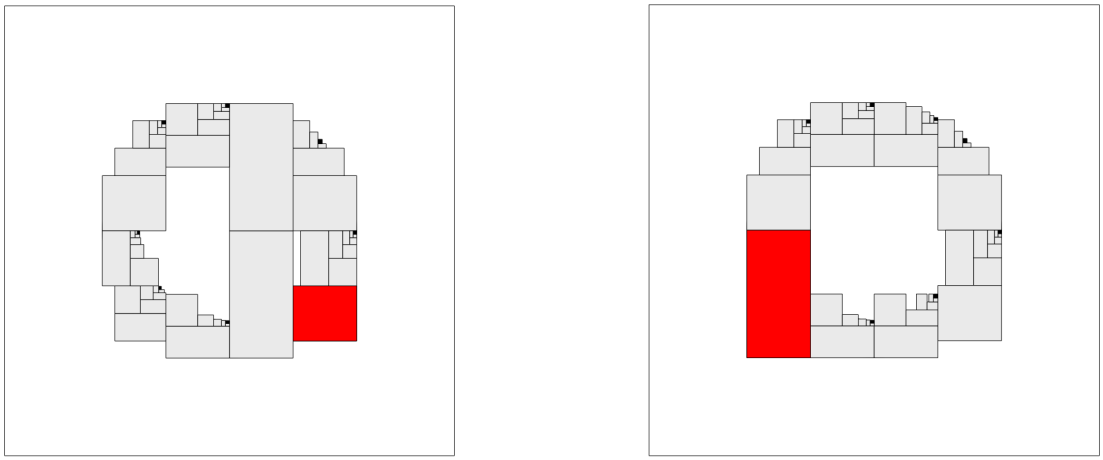
$$\rho_I([\mathbf{x}]) = \text{Profondeur}([\mathbf{x}]) \tag{3}$$

La profondeur dans l'arbre est une propriété statique de chaque boîte et IDFS suit le schéma de l'algorithme 3. L'évaluation de ρ_I a une complexité de $O(1)$ si cette information est mémorisée à chaque nœud de l'arbre de recherche.

4.2 Depth and Largest-First Search

Pour cette autre stratégie, nous exploitons l'hypothèse qui considère qu'entre deux boîtes non-explorées, celle de plus grande largeur contient probablement davantage de solutions. L'heuristique Largest-First (LF) traduit cette hypothèse et peut être évaluée par la fonction d'évaluation $\rho_L : \mathbb{I}^n \to \mathbb{R}$:

$$\rho_L([\mathbf{x}]) = w([\mathbf{x}]) \tag{4}$$



(a) DMDFS (b) DLFS

FIGURE 2 – Comportement des stratégies DMDFS et DLFS sur le problème $\mathcal{P}_1$ et identification de la prochaine boîte extraite après 7 solutions calculées.

La liste des boîtes non explorées est triée selon les valeurs de ρ_L. De la même façon que pour IDFS, l'implémentation de cette stratégie est une version statique (algorithme 3) de BestFS/DFS désignée par *Depth and Largest-First search* (DLFS). L'étape d'évaluation correspondant au calcul de ρ_L a une complexité en $O(n)$.

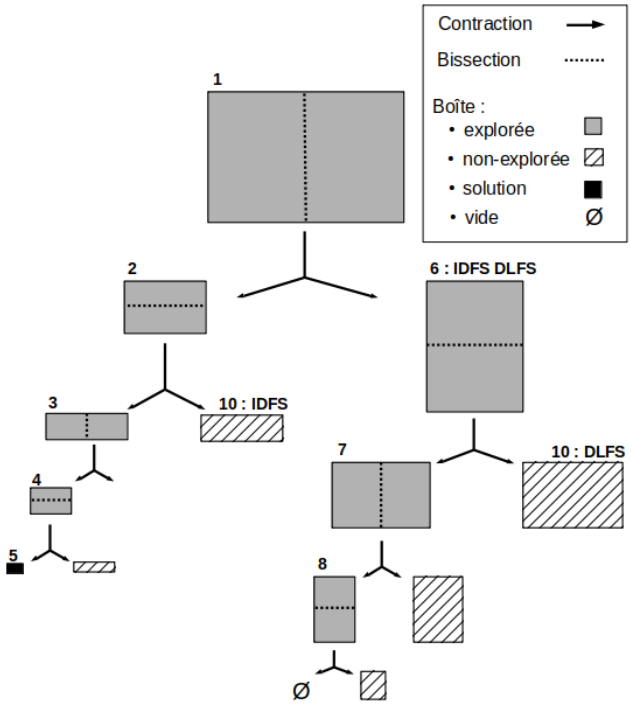


FIGURE 3 – Illustration de la différence entre IDFS et DLFS dans le parcours d'un arbre de recherche.

La figure 2 montre la différence entre DLFS et DMDFS lors de l'étape **ExtrairePremier** après 7 solutions trouvées sur le NCSP $\mathcal{P}_1$ ($\epsilon = 10^{-1}$). La figure 3 illustre la différence entre le comportement de DLFS et de IDFS lors de l'exploration d'un espace de recherche matérialisé par la boîte 1. Après le premier parcours en profondeur (1 à 5) les deux stratégies sélectionnent la boîte 6 qui est à la fois la boîte la plus large et celle de plus haut niveau dans l'arbre de recherche. Le second parcours en profondeur (6 à 9) donne une boîte vide. A ce stade, les deux stratégies divergent pour explorer la boîte la plus large pour DLFS et la boîte de plus haut niveau pour IDFS.

5 Évaluation expérimentale

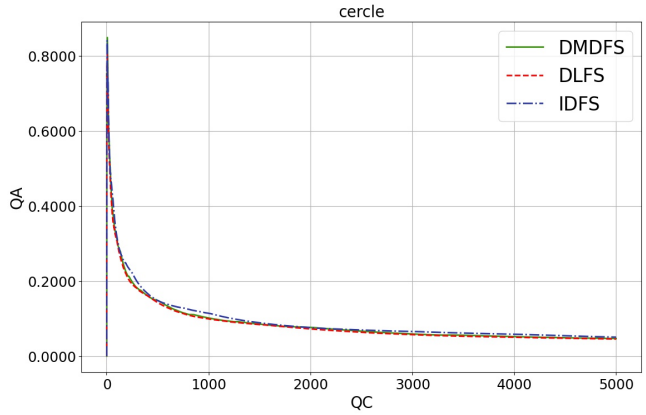
5.1 Indicateurs de qualité

Trois indicateurs de qualité sont introduits pour évaluer la performance des stratégies mises en place Soit $\Sigma = \{\sigma_1, ..., \sigma_m\}$ un ensemble de ϵ-boîtes en sortie d'un ABP. On définit :

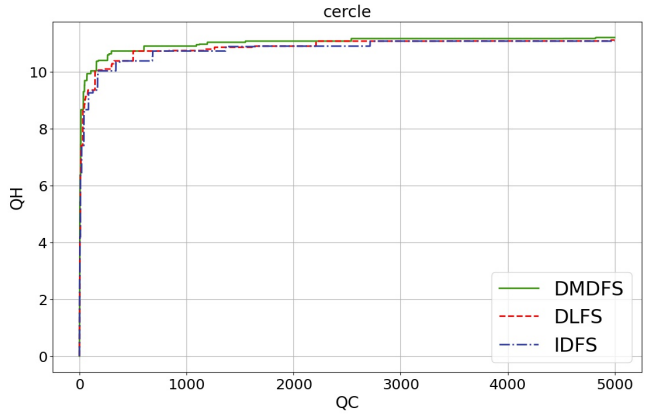
— Q_C la cardinalité m de Σ ;
— Q_H le volume de l'enveloppe $\square\Sigma$;
— Q_A la distance minimale moyenne entre les σ_k, soit :

$$Q_A = \frac{1}{m} \times \sum_{i=1}^{m} \min\{d_H(\sigma_i, \sigma_j) : 1 \leq j \leq m, j \neq i\}.$$

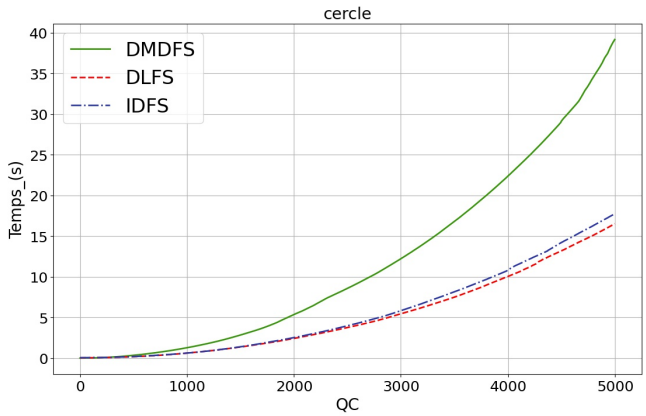
Ces indicateurs traduisent les propriétés attendues d'un algorithme de résolution anytime. Q_C représente le nombre de solutions calculées. Q_H décrit le degré de couverture de l'espace des solutions du NCSP. L'uniformité de la distribution des solutions calculées est évaluée par Q_A.



(a)



(b)



(c)

FIGURE 4 – Profils de performances des stratégies DMDFS, IDFS et DLFS en fonction de Q_C pour le problème simple $\mathcal{P}_1$ avec le scenario 0.

Nous verrons que Q_A ne peut-être comparé qu'entre deux sous-ensembles de même taille (Q_C équivalent).

5.2 Protocole

Les tests ont été réalisés sur une machine Linux Core i7-9850H 2.6 GHz (16 GB). Les algorithmes sont implémentés en C++ avec la librairie IBEX [1] basée sur l'arithmétique par intervalles. Les stratégies de contraction et de découpage des boîtes sont fixées : HC4, propagation de type AC3 et bissection de type RoundRobin (par ordre cyclique des variables).

Les NCSPs utilisés pour les tests sont tirés de la librairie GAMS [2]. Ce sont initialement des problèmes d'optimisation pour lesquels nous explorons uniquement l'espace réalisable sans tenir compte de l'objectif. Pour les problèmes non-contraints (cuspidal, ackley, rosenbrock), nous avons fixé un seuil sur l'objectif pour se ramener à un NCSP. Les tests ont été réalisés selon trois scenarii :

- Scenario 0 : nombre de solutions limité à $Q_C^{max} = 5000$, précision $\epsilon = 10^{-3}$;
- Scenario 1 : temps de résolution limité à t_{max}, précision $\epsilon = 10^{-3}$;
- Scenario 2 : nombre de solutions limité à $Q_C^{max} = 800$, précision $\epsilon = 10^{-3}$.

On distingue les problèmes dit *simples* dont les contraintes ne sont que des inéquations ($\mathcal{P}_1$, ackley, chance, cuspidal, ex7_2_10, rosenbrock), et les problèmes *complexes* qui comportent en plus des équations (bearing, alkyl, ship et himmel16).

5.3 Résultats

Les courbes de la figure 4 donnent l'évolution des indicateurs Q_H, Q_A et du temps de résolution en fonction de Q_C pour le NCSP $\mathcal{P}_1$ sur le scénario 0. Ce problème illustre les tendances des profils de performances pour les problèmes simples. On observe sur la figure 4c que la stratégie DMDFS nécessite des temps de calcul nettement plus importants pour calculer un même nombre de solutions. Ce résultat est une conséquence directe du coût de la fonction d'évaluation ρ_M par rapport aux stratégies statiques IDFS et DFLS.

La première partie des courbes 4a et 4b montrent une pente rapide suivie d'un plateau asymptotique. La qualité d'une approximation est donc fortement déterminée par les premières solutions calculées. On retrouve ce résultat sur l'ensemble des problèmes simples.

Le tableau 1 résume les résultats des scénarios 1 et 2 pour les problèmes simples avec : le nombre de variables (#V), le nombre de contraintes (#C). Les valeurs des indicateurs Q_A et Q_H ont été normalisées par $Q_i^{norm} = Q_i/Q_{max}$. Sur le scénario 1, t_{max} est déterminé comme le temps nécessaire à l'heuristique

1. http://www.ibex-lib.org/
2. https://www.gams.com/latest/gamslib_ml/libhtml/index.html#gamslib

				Scénario 1				Scénario 2		
Problème	**#V**	**#C**	**Stratégie**	**Temps (s)**	Q_C	Q_A	Q_H	**Temps (s)**	Q_A	Q_H
			DLFS	1,00	993	0,69	0,87	0,70	0,84	0,84
ackley_3	3	1	DMDFS	1,00	586	1,00	**1,00**	1,71	**1,00**	**1,00**
			IDFS	1,00	914	0,79	0,97	0,79	0,93	0,93
			DLFS	7,50	898	0,27	0,00	6,19	0,24	10^{-4}
ackley_12	12	1	DMDFS	7,50	616	0,22	0,06	12,21	0,25	0,07
			IDFS	7,51	855	1,00	**1,00**	6,77	**1,00**	**1,00**
			DLFS	20,01	834	0,01	0,00	19,11	0,01	10^{-4}
ackley_18	18	1	DMDFS	20,01	630	0,03	0,44	27,30	0,03	0,44
			IDFS	20,02	952	1,00	**1,00**	16,25	**1,00**	**1,00**
			DLFS	0,20	759	0,82	0,99	0,22	0,98	0,98
chance	4	3	DMDFS	0,20	459	1,00	0,99	0,57	**1,00**	**1,00**
			IDFS	0,20	902	0,51	**1,00**	0,16	0,66	0,99
			DLFS	1,00	792	0,83	0,88	1,02	0,94	0,88
cuspidal	3	1	DMDFS	1,00	541	1,00	0,94	2,32	**1,00**	0,94
			IDFS	1,00	882	0,73	**1,00**	0,83	0,87	**1,00**
			DLFS	5,00	786	0,73	0,65	5,19	0,79	0,66
ex7_2_10	11	9	DMDFS	5,00	431	1,00	**1,00**	18,22	**1,00**	**1,00**
			IDFS	5,00	829	0,68	0,65	4,66	0,75	0,66
			DLFS	0,30	673	0,77	0,94	0,42	0,95	0,94
rosenbrock_3	3	1	DMDFS	0,30	411	1,00	**1,00**	1,18	**1,00**	**1,00**
			IDFS	0,30	646	0,82	0,94	0,46	0,98	0,94
			DLFS	3,56	750	0,86	0,93	3,69	0,92	0,95
rosenbrock_6	6	1	DMDFS	3,50	580	1,00	**1,00**	4,68	**1,00**	**1,00**
			IDFS	3,50	699	0,93	0,89	3,81	0,99	0,87
			DLFS	-	811	0,62	0,66	4.57	0,71	0,66
Valeurs moyennes	-	-	DMDFS	-	532	0,78	0,81	8.53	0,78	0,81
			IDFS	-	835	0,81	0,93	4.22	0,90	0,92

Tableau 1 – Résultats normalisés des expérimentations sur les problèmes simples (inéquations uniquement). Scénario 1 : temps de résolution limité à t_{max}. Scénario 2 : nombre de solutions limité à $Q_C^{max} = 800$.

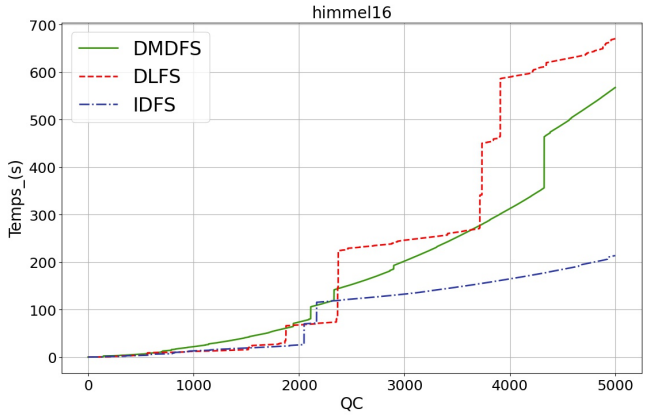
la plus performante pour atteindre le plateau asymptotique. En dehors de ackley_12 et ackley_18 DMDFS obtient de meilleurs résultats mais calcule moins de solutions (35% de moins en moyenne). Rapporté au temps de calcul, IDFS obtient en moyenne de meilleurs résultats sur les trois indicateurs.

L'analyse sur Q_A est moins immédiate. En effet cet indicateur décroît avec le nombre de solutions trouvées et nous cherchons l'approximation le maximisant. De fait, entre deux sous-ensembles de solutions, celui avec le Q_C le plus grand donnera, sauf exception, le Q_A le plus faible. Ainsi, on ne comparera Q_A qu'à Q_C équivalent et le scénario 1 ne permet pas de conclure sur l'indicateur Q_A.

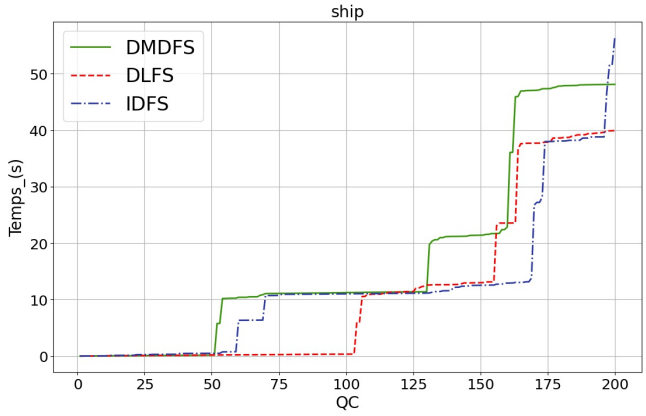
Les tests du scénario 2 à $Q_C = 800$ montrent de meilleures performances pour DMDFS sur Q_A sauf pour les problèmes ackley_12 et ackley_18. Ce résultat est attendu puisque la fonction d'évaluation de DMDFS revient à maximiser Q_A au fur et à mesure de l'exploration. Sur l'ensemble des tests IDFS est plus performant avec en moyenne $Q_A = 0,9$. Pour l'indicateur Q_H DMDFS obtient la meilleure approximation sur 5 problèmes parmi les 8. IDFS est cependant plus efficace pour obtenir rapidement ces approximations.

Les courbes de la figure 5 illustrent les profils de performances pour les problèmes complexes. On observe que les temps de calcul dépendent énormément du problème choisi. Par exemple pour himmel16, DLFS est la stratégie la moins efficace, pour le problème alkyl il s'agit de IDFS. Au niveau des indicateurs les résultats sont disparates, on ne peut pas conclure ici. Cette observation peut être expliquée par la génération importante de boîtes non explorées lors de certains backtracks, augmentant la taille de l'arbre de recherche. De plus, une grande partie de ces boîtes vont donner des boîtes vides à l'issue de la contraction du fait des contraintes d'égalité.

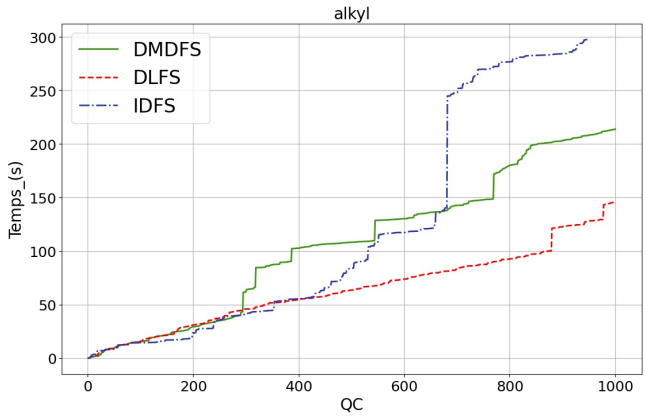
Finalement, l'heuristique dynamique DMDFS présente le meilleur comportement Anytime pour des problèmes simples sans contraintes d'égalité et à nombre de solutions équivalent. Néanmoins ce type de stratégie peut conduire rapidement à des temps de calculs impor-



(a)



(b)



(c)

FIGURE 5 – Temps de calcul des stratégies DMDFS, IDFS et DLFS en fonction du nombre de solutions obtenue pour trois problèmes complexes avec une précision de 10^{-3}.

tants dès que l'on augmente le nombre de ϵ-boîtes dans l'approximation attendue. Sur l'ensemble de l'étude, IDFS représente le meilleur compromis entre la qualité de l'approximation (Q_H et Q_A), le temps de calcul et un nombre de solutions suffisamment grand (Q_C). Dans les conditions de notre étude et pour des NCSP non-linéaires complexes les performances des stratégies ne sont pas uniformes et dépendent largement du problème choisi.

6 Conclusion

Ce travail propose d'élargir le domaine des problèmes de satisfaction de contraintes numériques à une approche de résolution partielle, dite Anytime, déjà utilisée en optimisation. Ce nouveau paradigme demande la construction de stratégies de recherche spécifiques pour calculer des approximations représentatives de l'espace des solutions. Des indicateurs de qualité sont proposés pour évaluer ces stratégies entre elles. Q_H et Q_A traduisent l'exploration diversifiée (couverture et uniformité) de l'espace des solutions.

Deux nouvelles stratégies de recherches pour algorithmes Branch-and-Prune Anytime sont proposées : IDFS et DLFS. Elles sont comparées à la stratégie existante DMDFS. Cet article a permis de distinguer deux comportements : dynamique et statique. Cette différence se traduit par une complexité temporelle plus importante pour les stratégies dynamiques comme DMDFS. Si le temps de calcul n'est pas considéré, DMDFS obtient de meilleures évaluations pour les indicateurs Q_H et Q_A à Q_C (nombre de solutions) fixé. Dans le cas d'un nombre restreint de solutions, DMDFS peut être préférable si la taille de l'arbre de recherche ne croît pas trop vite (difficilement prévisible). L'évaluation de la complexité et les résultats observés sur un ensemble de problèmes simples, sans contrainte d'égalité, montrent les propriétés supérieures de la stratégie IDFS.

En revanche, les expérimentations ne permettent pas de conclure sur l'exploration anytime de problèmes plus complexes. Plusieurs pistes peuvent être envisagées pour attaquer ce point, notamment l'impact des étapes de division des domaines et de l'étape de tri des boîtes non-explorées pour ces stratégies BestFS. Il sera aussi pertinent d'adapter la phase de descente en profondeur DFS pour contrôler davantage l'exploration vers des solutions diversifiées.

Parmi les perspectives à court terme, l'étude de ces stratégies sur des problèmes mixtes permettra de traiter des problèmes issues de la conception préliminaire. IDFS et DLFS sont de bons candidats car la profondeur ne s'appuie pas sur les domaines des variables et la notion de largeur peut-être aisément étendue aux entiers.

Les indicateurs de comparaison proposés ici se focalisent principalement sur les ϵ-boîtes et pourraient aussi s'appliquer aux différents nœuds de l'arbre de recherche.

Dès lors, nous pouvons définir des usages beaucoup plus interactifs des solveurs. Tous ces éléments ont pour objectif, in fine, de permettre une utilisation plus active des outils d'exploration de concepts.

Références

[1] F. Benhamou, F. Goualard, L. Granvilliers et J-F Puget : Revising hull and box consistency. *International Conference on Logic Programming*, pages 230–244, 01 1999.

[2] P. Chenouard, R.and Sébastian et L. Granvilliers : Solving an Air Conditioning System Problem in an Embodiment Design Context Using Constraint Satisfaction Techniques. 4741/2007:18–32, septembre 2007.

[3] R. Chenouard, A. Goldsztejn et C. Jermann : Search strategies for an anytime usage of the branch and prune algorithm. *IJCAI International Joint Conference on Artificial Intelligence*, 01 2009.

[4] R. Dechter et J. Pearl : Generalized best-first search strategies and the optimality of a*. *Journal of the ACM*, 32(3), 1985.

[5] Merlet J-P., Chablat D., Wenger P. et Majou F. : An interval analysis based study for the design and the comparison of three-degrees-of-freedom parallel kinematic machines. *Int. J. Robotics Res.*, 23(6):615–624, 2004.

[6] L. Jaulin, O. Kieffer, M.and Didrit et E. Walter : *Applied Interval Analysis*. Springer London Ltd, août 2001.

[7] A. Jesus, L. Paquete et A. Liefooghe : A model of anytime algorithm performance for bi-objective optimization. *Journal of Global Optimization*, 05 2020.

[8] Olivier Lhomme : Consistency techniques for numeric csps. pages 232–238, 01 1993.

[9] M. Li et X. Yao : Quality evaluation of solution sets in multiobjective optimisation : A survey. *ACM Computing Surveys*, 52:1–38, 03 2019.

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

[11] P. Meseguer : Interleaved depth-first search. *In Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence - Volume 2*, IJCAI'97, page 1382–1387, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[12] D. R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe et Edward C. Sewell : Branch-and-bound algorithms : A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79 – 102, 2016.

[13] B. Neveu, G. Trombettoni et I. Araya : Node selection strategies in interval Branch and Bound algorithms. *Journal of Global Optimization*, 64(2): 289–304, février 2016.

[14] S. Sayin : Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. *Mathematical Programming, Series B*, 87:543–560, 05 2000.

[15] G. Trombettoni et G. Chabert : Constructive Interval Disjunction. *In CP'07 - 13th International Conference on Principles and Practice of Constraint Programming*, 2007.

[16] P. Van Hentenryck, D. Mcallester et D. Kapur : Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34, 07 2001.