UNIVERSITÉ Clermont Auvergne

ISIMA
Institut Supérieur d'Informatique de Modélisation et de leurs Applications

Rapport de projet

Étude et réduction du temps de calcul d'un algorithme d'optimisation utilisant l'analyse par intervalles

Réalisé par :
Salma LAMYAGHRI, Aicha RIDOUAN
Encadré par :
M. Sebastien LENGAGNE

Filière :
Génie logiciel et systèmes d'information

Année universitaire :
2020-2021

Table des matières

Remerciement **6**

Résumé **7**

Abstract **8**

Introduction **9**

1 Présentation des problèmes d'optimisation sous contraintes **10**
- 1.1 Contexte général 10
- 1.2 Problèmes sous contraintes 10
  - 1.2.1 Problèmes de Satisfaction de Contraintes 10
  - 1.2.2 Problèmes d'Optimisation de Contraintes 11
- 1.3 Analyse par intervalles 11
  - 1.3.1 Définition des intervalles 11
  - 1.3.2 Définition des boîtes 12
  - 1.3.3 Fonctions d'inclusion 12
  - 1.3.4 Définition du pessimisme 13
- 1.4 Définition de la problématique 13

2 Résolution des problèmes sous contraintes à l'aide de l'analyse par intervalles **14**
- 2.1 Algorithme de Bissection 14
  - 2.1.1 Principe de l'algorithme 14
  - 2.1.2 Mise en oeuvre de la bissection 15
- 2.2 Fonctions BSplines 16
  - 2.2.1 Définition formelle 16
  - 2.2.2 Calcul des fonctions de base 16
  - 2.2.3 Calcul des points de contrôle 17
  - 2.2.4 BSplines et réduction du pessimisme 17
- 2.3 Combinaison de la Bissection et des fonctions BSplines pour la résolution des POC 18

3 Optimisation du calcul des Bsplines **19**
- 3.1 Motivations 19
- 3.2 Définition de la complexité 19
- 3.3 Première méthode : Mise à jour de la matrice des fonctions de base 20
  - 3.3.1 Principe de la méthode 20
  - 3.3.2 Pseudo-code 21

3.4 Normalisation et mise à jour des coefficients du polynôme . . . 22
3.4.1 Principe de la méthode . . . 22
3.4.2 Pseudo-code . . . 23
3.5 Comparaison des méthodes et Conclusion . . . 24

4 Mise en oeuvre et résultats **25**
4.1 Résultats et Interprétations . . . 25
4.1.1 Langages de programmations utilisés . . . 25
4.1.2 Mise en oeuvre des résultats . . . 25
4.2 Gestion du projet . . . 26
4.2.1 Organisation de l'équipe du projet . . . 26
4.2.2 Intégration continue . . . 26
4.2.3 Réunions hebdomadaires . . . 26
4.2.4 Déroulement du travail . . . 26
4.2.5 Diagramme de Gantt . . . 27

Conclusion **28**

A La bissection **29**

Table des figures

1.1 Réprésentation d'une boite à deux dimensions 12
1.2 Représentation des deux fonctions d'inclusion [m] et $[m]^*$ 12

2.1 Résultats de l'implémentation de la bissection 16
2.2 Résultats de l'implémentation des BSplines 18

4.1 Résumé des données du problème 25
4.2 Résultats de la bissection 26
4.3 Diagramme de Gantt 27

A.1 Application de l'algorithme de bissection sur 2 dimensions 30

Remerciement

Tout d'abord, on tient à remercier tout le corps professoral et admnistratif de l'Institut d'Informatique d'Auvergne ISIMA, pour la richesse et la qualité de leur enseignement et qui déploient de grands efforts pour assurerà leurs étudiants une formation actualisée, et plus particulièrement à M. LOIC Yon responsable de la filière Génie logiciel et systèmes d'information.
On tient aussi à remercier sincèrement M. LENGAGNE Sébastien qui est à l'écoute tout au long des réunions et qui nous a accompagné durant les phases de réalisation de ce projet. Nous tenons à le remercier ainsi pour l'inspiration, l'aide et le temps qu'il a bien voulu nous concacrer.
Enfin, nos remerciements s'étendent à toute personne ayant contribué de près ou de loin dans l'élaboration de notre projet.

Résumé

Le but de ce projet est de faire une étude d' un algorithme d'optimisation existant afin de proposer des méthodes pour réduire le temps de calcul. L'implémentation a été réalisée en deux langages de programmation, C++ et Python, avec l'éditeur de texte Visual Studio Code dans un environnement Linux et en utilisant CMake pour gérer la compilation. Pour l'intégration continue de notre code, la plateforme Forge a été utilisée et plus précisément le dépôt git intégré dedans en travaillant dans deux branches différentes, chacune pour un langage et en intégrant les deux codes dans la branche master. Le travail réalisé dans ce projet serait probablement le point de départ d'un projet à venir, en effet notre travail à présenté plusieurs pistes d'optimisation pour cette algorithme et a relevé plusieurs failles relatives notamment aux problème de troncature des nombres réels.

Mots-clés : Algorithm d'optimisation, C++, Python, Visual Studio Code, CMake, Forge, Git, Branche

Abstract

The topic of this project was about an optimization algorithm. The goal of our study is to suggest some methods that can reduce calculation time of this algorithm. The implementation was achieved using two programming languages, C++ and Python, carried out under the text editor Visual Studio Code in Linux environment, and for managing the compilation process, CMake was used. For continuous integration of our code, the git repository provided by the Forge platform has been used, working with 2 branches, one for each programming language, plus the master branch where the pull were made. This work is a starting point of a next coming project, in fact,our work represent multiple ways of optimization for this algorithm and throw away several problems that are related to floating numbers.

Keywords : Optimization algorithm, C++, Python, Visual Studio Code, CMake, Forge, Git, Branch

Introduction

La résolution des problèmes de statisfaction de contraintes consiste à trouver l'ensemble de toutes les variables qui satisfont les contraintes du problème. Ces problèmes peuvent être étendus aux problèmes d'optimisation de contraintes qui eux cherchent à trouver les variables qui donnent la meilleure solution aux problème.
L'objet de ce projet est l'étude d'un algorithme d'optimisation sous contraintes à l'aide de l'analyse par intervalles, qui s'est avéré très efficace dans ce contexte, notamment pour la rigueur et la précision de ses résultats. L'algorithme étudié a été conçu en utilisant une combinaison deux outils mathémtiques et algorithmiques :

- La bissection : Il s'aigit d'une méthode dichotomique qui nous permettra de borner la solution optimale avec une grande précision.
- Les fonctions BSplines : Qui nous permettront de réduire le problème de pessimisme qu'on rencontre souvent en analyse par intervalle.

Notre mission principale tout au long de ce projet sera d'améliorer l'implémentation de cet algorithme afin d'optimiser son temps de calcul.

Chapitre 1

Présentation des problèmes d'optimisation sous contraintes

Dans cette première partie, nous allons définir la problématique traitée tout au long de ce projet ainsi que les éléments de recherche sur lesquels nous avons basé notre travail afin de cerner le sujet et de nous positionner dans un contexte scientifique bien précis.

1.1 Contexte général

Les problèmes de satisfaction de contraintes et les problèmes d'optimisation sous contraintes ont de nombreuses applications dans différents domaines scientifiques et notammant en Robotique, tel que la plannification du mouvement d'un robot **[1]**.
Ce types de problèmes fait l'objet de plusieurs recherches en intelligence artificielle et recherche opérationnelle, et beaucoup de méthodes de résolution on été développées à l'aide des heuristiques ou de l'optimisation combinatoire.
L'analyse par intervalle (AI) est un outil mathématique très puissant et qui s'est avéré efficace dans la résolution des problèmes de satisfaction de contraintes. En effet l'AI garantit toujours un optimum global tout en gardant un temps d'exécution raisonnable.

1.2 Problèmes sous contraintes

1.2.1 Problèmes de Satisfaction de Contraintes

Les problèmes de satisfaction de contraintes (PSC) sont des problèmes mathématiques qui consistent à trouver un ensemble de variables respectant les contraintes définies dans le problème.

Définition formelle :

Trouver tous les $[q] \in \mathbb{Q}$ tels que :

$$\forall j \in \{1, ..., m\} \ \varphi([q]) \in [\underline{g}_j, \bar{g}_j] \tag{1.1}$$

Avec :

- n est le nombre de variables de l'ensemble q
- m est le nombre de contraintes
- q $= \{q_1, ..., q_n\}$ est un ensemble de n variables
- $\mathbb{Q} = \{[\underline{q}_1, \bar{q}_1], ..., [\underline{q}_n, \bar{q}_n]\} \subset \mathbb{R}^{\ltimes}$
- $\varphi([q])$ est un ensemble de m contraintes. Chaque $\varphi_j([q])$ doit être inclus dans l'intervalle $[\underline{g}_j, \bar{g}_j]$

1.2.2 Problèmes d'Optimisation de Contraintes

Comme les PSC se focalisent sur les solutions valides mais pas nécessairement optimales, il a été nécessaire d'introduire les problèmes d'optimisation sous contraintes (POC) qui - eux - consistent à trouver l'ensemble des variables qui non seulement respectent les contraintes du problèmes mais aussi optimisent les fonctions critère du prblème.

Définition formelle :

Trouver tous les $[q] \in \mathbb{Q}$ tels que :

$$\underset{q}{min}\ \mathcal{F}([q]) \tag{1.2}$$

Avec

$$\forall j \in \{1, ..., m\}\ \varphi([q]) \in [\underline{g}_j, \bar{g}_j]$$

Où $\mathcal{F}([q])$ est la fonction critère

1.3 Analyse par intervalles

1.3.1 Définition des intervalles

Un intervalle $[q] = [\underline{q}, \bar{q}]$ est un sous ensemble connecté et fermé de $\mathbb{R}$ avec $\underline{q} = inf([q])$ et $\bar{q} = sup([q])$.
Soient [a] et [b] deux intervalles, on peut définir les opérateurs arithmétiques entre [a] et [b] comme suit **[5]** :

$$[a] + [b] = [\underline{a} + \underline{b}, \bar{a} + \bar{b}] \tag{1.3}$$

$$[a] - [b] = [\underline{a} + \bar{b}, \bar{a} + \underline{b}] \tag{1.4}$$

$$[a] * [b] = [min(\underline{a}\underline{b}, \underline{a}\bar{b}, \bar{a}\underline{b}, \bar{a}\bar{b}), max(\underline{a}\underline{b}, \underline{a}\bar{b}, \bar{a}\underline{b}, \bar{a}\bar{b})] \tag{1.5}$$

$$[a]/[b] = [a] * [1/b]\ si\ 0 \notin [b] \tag{1.6}$$

$$Si\ f \in \{cos, sin, sqr, sqr, log, ...\}\ f([a]) = [f(a)|a \in [a]] \tag{1.7}$$

1.3.2 Définition des boîtes

On définit les boîtes ou vecteurs d'intervalles par le sous ensemble de $\mathbb{R}$ qui représente le produit cartésien de plusieurs intervalles. Soit $[a]$ une boîte de dimension n, on a :

$$[a] = [a_1] \times [a_2] \times ... \times [a_n] \tag{1.8}$$

Où $[a_i] = [\underline{a}_i, \bar{a}_i]$ est la projection de [a] sur le $i^{\text{ème}}$ axe, $\forall i \in [\![1, n]\!]$.

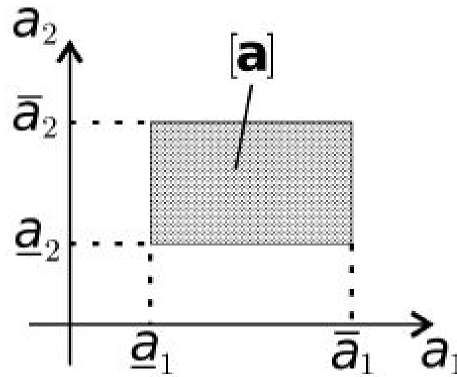

FIGURE 1.1 – Réprésentation d'une boite à deux dimensions

1.3.3 Fonctions d'inclusion

Définition formelle

On considère la fonction $\mathbf{m} : \mathbb{R}^n \longmapsto \mathbb{R}^m$; l'image de la boite [a] par cette fonction est :

$$\mathbf{m}([a]) = \{\mathbf{m}(u) \quad | \quad u \in [a]\} \tag{1.9}$$

$[\mathbf{m}] : \mathbb{R}^n \longmapsto \mathbb{R}^m$ est une fonction d'inclusion de $\mathbf{m}$ si :

$$\forall [a] \in \mathbb{R}^n, \mathbf{m}([a]) \subseteq [\mathbf{m}]([a]) \tag{1.10}$$

La figure ci-dessous illustre l'exemple d'une fonction d'inclusion $\mathbf{m} : \mathbb{R}^2 \longmapsto \mathbb{R}^2$ avec 2 intervalles $a_1 = [\underline{a}_1, \bar{a}_1]$ et $a_2 = [\underline{a}_2, \bar{a}_2]$

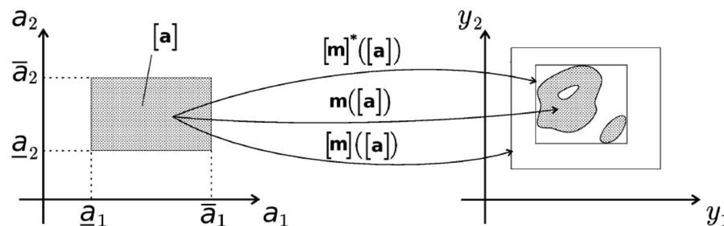

FIGURE 1.2 – Représentation des deux fonctions d'inclusion [m] et $[m]^*$

$m([a])$ est l'image de la boite $[a]$.
La fonction d'inclusion minimale notée $[m]^*$, est définie par :

$$\forall [m] \text{ fonction d'inclusion de m, } [m]^*([a]) \subseteq [m]([a]) \quad (1.11)$$

Il existe plusieurs types de fonctions d'inclusion et pour des raisons de simplicité, nous allons utiliser les fonctions d'inclusion naturelles. L'utilisation de ces fonctions consiste à remplacer chaque occurence d'une variable réelle par son intervalle.

1.3.4 Définition du pessimisme

Le pessimisme désigne une surestimation de l'image d'un intervalle par sa fonction d'inclusion, on se retrouve alors avec des intervalles plus larges que l'image réelle.
Exemple :
On prend les expressions suivantes de la même fonction f : $\mathbb{R} \longmapsto \mathbb{R}$:

$$f_1(a) = a(a+1)$$

$$f_2(a) = a * a + a$$

$$f_3(a) = a^2 + a$$

$$f_4(a) = (a + \frac{1}{2})^2 - \frac{1}{4}$$

Les images calculées par chacunes des fonctions d'inclusion naturelles ci-dessus pour l'intervalle a = [-1, 1] sont :

$$[f_1]([a]) = [-2, 2]$$

$$[f_2]([a]) = [-2, 2]$$

$$[f_3]([a]) = [-1, 2]$$

$$[f_4]([a]) = [\frac{-1}{4}, 2]$$

Dans cet exemple, les résultats diffèrent selon la formule mathématique de la fonction d'inclusion et plus précisement, selon le nombre d'occurences de la variable q dans la formule. On remarque que le pessimisme est plus accentué avec les expressions développées des fonctions d'inclusion. Tous les résultats précédents sont correctes, mais les résultats sont d'autant moins précis que le pessimisme est important.

1.4 Définition de la problématique

Il s'agit de réduire le pessimisme dans l'algorithme d'optimsation sous contraintes proposé dans la thèse de Rawan Kalawoun **[1]**, afin d'améliorer les performances de l'algorithme en termes de temps d'exécution.

Chapitre 2

Résolution des problèmes sous contraintes à l'aide de l'analyse par intervalles

Dans cette partie, nous allons faire une description algorithmique des différentes méthodes que nous allons utiliser pour résoudre les problèmes d'optimisation sous contraintes à l'aide de l'analyse par intervalles.

2.1 Algorithme de Bissection

2.1.1 Principe de l'algorithme

La bissection est une méthode itérative qui consiste à décomposer l'ensemble d'entrée en plusieurs ensembles ayant des tailles de plus en plus petitess.
À chaque itération de l'algorithme, la boîte courante est évaluée par rapport aux contraintes et à la fonction critère. Selon les résultats de cette évaluation, la boîte sera soit rejetée ou découpé et empilée, en effet :

- Une boite est rejetée si elle ne respecte pas l'une des contraintes définies dans le problème

- Sinon, elle sera découpée en deux parties et empilée pour être traitée dans une prochaine itération

La répartition de la boite permet une évaluation plus précise de la fonction critère et des contraintes. Le processus s'arrête lorsque la taille de la boite est plus petite que le seuil défini dans les paramètres.

Les étapes expliquées ci-dessus sont mises en oeuvre dans l'algorithme suivant :

Algorithm 1: Bissection

Entrée: Q l'espace de recherche initial
ϵ : la présicion désirée
Sortie : Le boite optimale $\tilde{q}$
Initialisation Q.empiler($\mathbf{Q}$), $\tilde{f} = \infty$
while Q *non vide* **do**
  [q] = Q.dépiler()
  $[f] = [\underline{f}, \bar{f}] = \mathcal{F}[q]$
  if $\underline{f} < \tilde{f}$ **then**
    Evaluer les contraintes : $\forall j [g_j] = \mathcal{G}_j([q])$
    if $\forall j [g_j] \cap [\underline{g}, \bar{g}] = \emptyset$ **then**
      if $[g_j] \cap [\underline{g}, \bar{g}] = [g_j] and \bar{f} < \tilde{f}$ **then**
        $\tilde{f} = \bar{f}$
        $\tilde{q} = q$
      end if
      if $diam([q]) > \epsilon$ **then**
        $\{\mathrm{q}_1, q_2\} = bissection(q)$
        Q.empiler(q_1)
        Q.empiler(q_2)
      end if
    end if
  end if
end while
return $\tilde{q}$

2.1.2 Mise en oeuvre de la bissection

La figure suivante met en oeuvre une implémentation de la bissection avec des boîtes bi-dimensionnelles.

- Les boîtes rouges représentent les boîtes rejetées
- Les boîtes bleu représentent les boîtes acceptées
- Les boîtes vertes représentent les boîtes qui contiennent potentiellemnt des intervalles acceptés, il s'agit de la catégorie semi-sacceptée.

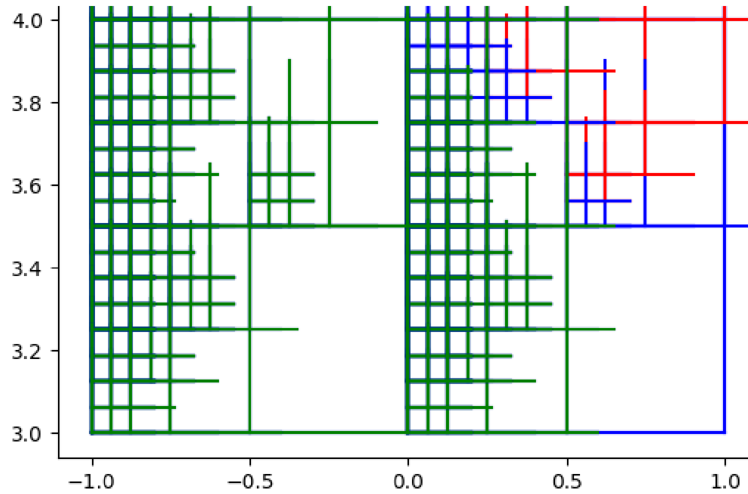

FIGURE 2.1 – Résultats de l'implémentation de la bissection

2.2 Fonctions BSplines

2.2.1 Définition formelle

Une fonction BSpline de degré n est la somme pondérée de plusieurs fonctions de base de degré n. Elle est définie par m points de contrôle comme suit :

$$F(q) = \sum_{i=1}^{m} B_{i,n}(q) \times P_i \tag{2.1}$$

Avec $\forall q \in [\underline{q}, \bar{q}]$:

$$\sum_{i=1}^{m} B_{i,n}(q) = 1 \tag{2.2}$$

En calculant le minimum et le maximum des points de contrôle et en utilisant la propriété (2.2), on obtient directement une estimation des bornes de F(q). En effet :

$$\forall i \in [\![1, m]\!], \forall q \in [\underline{q}, \bar{q}] : \underline{P} \leq P_i \leq \bar{P} \implies \underline{P} \leq F(q) \times P_i \leq \bar{P} \tag{2.3}$$

2.2.2 Calcul des fonctions de base

Pour le calcul des fonctions de base, nous avons utilisé la formule de Cox-Deboor **[2]** qui propose une méthode récursive pour l'évaluation des bsplines comme suit :
$\forall j \in [\![0, m-n-1]\!]$:

$$B_{j,0}(t) := \begin{cases} 1 \ si \ t_j \leq t \leq t_{j+1} \\ 0 \ sinon \end{cases} \tag{2.4}$$

$$B_{j,n}(t) := \frac{t - t_j}{t_{j+n} - t_j} B_{j,n-1}(t) + \frac{t_{j+n+1} - t}{t_{j+n+1} - t_{j+1}} B_{j+1,n-1}(t). \tag{2.5}$$

Avec $\{t_1, t_2, \ldots, t_{m_n}\}$ sont des points dans l'intervalle $[q] = [\underline{q}, \bar{q}]$ et $n \leq m$.
Au niveau de l'implémentation, la programmation dynamique **[3]** a été utilisée au lieu de la récursion pour optimiser le temps de calcul.

2.2.3 Calcul des points de contrôle

On suppose que les fonctions utilisées sont toutes polynômiales, on a alors :

$$F(q) = [1, q, \ldots, q^n] \times [a_0, a_1, \ldots, a_n]^T \tag{2.6}$$

Par l'équation (2.1), on a :

$$F(q) = [1, q, \ldots, q^n] \times B \times [p_0, p_1, \ldots, p_n]^T \tag{2.7}$$

D'où :

$$[p_0, p_1, \ldots, p_n]^T = B^{-1} \times [a_0, a_1, \ldots, a_n]^T \tag{2.8}$$

Avec :

- $\{a_0, a_1, \ldots, a_n\}$ sont les coefficients du polynôme.
- B est la matrice contenant les coefficients des fonctions de base.
- $\{p_0, p_1, \ldots, p_n\}$ sont les points de contrôle.

Dans le cas multidimensionnel, la matrice B peut être écrite comme suit :

$$B = B_1 \otimes B_2 \otimes B_3 \otimes \cdots \otimes B_p \tag{2.9}$$

Où :

- p est la dimension de la fonciton F soit le nombre de variables de la fonciton F
- $\forall i \in [\![1, p]\!]$, B_i est la matrice des fonctions de base de F suivant la variable q_i
- $\otimes$ est l'opérateur du produit de Kronecker

On pose $P = [p_0, p_1, \ldots, p_n]^T$ et $X = [a_0, a_1, \ldots, a_n]^T$, on a :

$$P = (B_1 \otimes B_2 \otimes B_3 \otimes \cdots \otimes B_p)^{-1} \times X \tag{2.10}$$

En utilisant la propriété de commutativité de l'inverse d'un produit de Kronecker (i.e. : $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$) **[4]**, l'équation (2.10) devient :

$$P = (B_1^{-1} \otimes B_2^{-1} \otimes B_3^{-1} \otimes \cdots \otimes B_p^{-1}) \times X \tag{2.11}$$

L'équation (2.11) sera utilisée au niveau de l'implémentation pour optimiser les calcul.

2.2.4 BSplines et réduction du pessimisme

Le but derrière l'utilisation des BSplines est principalement la réduction du pessimisme. Les résultats ci-dessous mettent en oeuvre cet aspect et présentent une comparaison entre les images des intervalles calculés à l'aide de la fonction d'inclusion naturelle et celle calculée à l'aide des fonctions BSpline. La fonction considérée pour ce test est définie par $f(a, b) = 1 - 3a - 2b - 4ab$ et le vecteur d'intervalles considéré est $b = [-1; 1] \times [-1; 1]$

	Intervalle Image
Fonction naturelle	[-4 ; 10]
BSplines	[-8 ; 10]

Figure 2.2 – Résultats de l'implémentation des BSplines

2.3 Combinaison de la Bissection et des fonctions BSplines pour la résolution des POC

L'algorithme d'optimisation sous contraintes étudié dans ce projet est basé sur une méthode qui combine les BSplines et la bissection. Les fonctions bsplines sont utilisées pour estimer les bornes de la fonction d'inclusion à partir du min et du max des points de contrôles, tout en réduisant le pessimisme. La bissection permet ensuite de trouver l'optimum de la fonction critère en respectant les fonctions contrainte du problème. L'algorithme est donc composé de deux phases principales :

- La première étant celle de la préparation de la matrice des fonctions de base qui servira au calcul des points de contrôle.
- La deuxième phase est celle de la bissection dans laquelle un intervalle est divisé en deux à chaque itération, ce qui nécessite des mises à jours des données permetant le calcul des Bsplines.

Comme la phase de préparation s'est avérée très couteuse en terme de temps d'exécution, il a fallu modifier le processus de mise à jour utilisé dans la deuxième phase, sans avoir besoin de recalculer la matrice des fonctions de base afin de minimser le temps de calcul. Ces éléments seront détaillés dans la partie suivante.

Chapitre 3

Optimisation du calcul des Bsplines

Dans ce chapitre nous allons étudier la complexité de l'algorithme permettant le calcul des BSplines en utilisant deux méthodes différentes afin d'améliorer la performance du code.

3.1 Motivations

Dans l'algorithme étudié dans ce projet, les fonctions BSplies sont utilisées en tant que fonctions d'inclusion. L'image du vecteur d'intervalles est alors estimée à partir des bornes sup et inf des points de contrôles.
L'expression de la matrice des points de contrôle est donnée par l'equation (2.9) et (2.10) :

$$P = B^{-1} \times X$$

Le calcul de la matrice P peut se faire de deux méthodes :

- La première méthode consiste à garder la matrice B constante et varier le vecteur X en fonction des bornes des compostants du vecteur d'intervalles passée en entrée aux BSplines
- La deuxième méthode consiste à garder le vecteur X constant et varier la matrice B en fonction des bornes des composants du vecteur d'intervalles

Dans ce chapitre nous allons étudier en détails chacune de ces deux méthodes et comparer la complexité temporelle de leurs implémentations afin de choisir la méthode la plus optimale pour notre algorithme.

3.2 Définition de la complexité

En informatique la complexité temporelle est un moyen permettant de mesurer la durée d'exécution d'un algorithme. Elle correspond au nombre d'opérations élémentaires effectuées dans l'algorithme et dépend de la taille des entrées de l'algorithme.
Pour exprimer la complexité temporelle, on utilise les notations en Grand O : $O(f(n))$ où n représente la taille des entrées et f est une fonction quelconque.
Il existe plusieurs types de complexité selon la nature de la fonction f.

Exemples :

- Complexité constante : $O(1)$
- Complexité polynômiale : $O(n^k)$ où $k \in \mathbf{N}$
- Complexité logarithmique : $O(log(n))$
- Complexité exponentielle : $O(k^n)$ où $k \in \mathbf{N}$

3.3 Première méthode : Mise à jour de la matrice des fonctions de base

3.3.1 Principe de la méthode

Étant donnée la structure de l'algorithme de bissection utilisé, un seul intervalle parmi les composants du vecteur d'intervalles passé en entrée est modifié à chaque itération. En effet, par le principe de bissection, seul l'intervalle le plus large est divisé par deux à chaque itération. Il suffit donc de recalculer la matrice des BSplines lui correspondant pour mettre à jour la matrice principale des fonctions de base, qui représente le produit de Kronecker de toutes les matrices des fonctions de base. Le processus de mise à jour sera expliqué dans ce qui suit.
Soient $A \in M_n(\mathbf{R})$ et $B \in M_p(\mathbf{R})$, on pose :

$$A = (a_{i,j})_{1\leq i,j\leq n},\ B = (b_{i,j})_{1\leq i,j\leq p} \text{ et } D = A \otimes B = (d_{i,j})_{1\leq i,j\leq n\times p}$$

on a par définition du produit de Kronecker :
$\forall i, j \in [\![1,\ n \times p]\!], \exists l, m \in [\![1, n]\!]$ et $u, v \in [\![1, p]\!]$ tels que :

$$d_{i,j} = a_{l,m} \times b_{u,v} \tag{3.1}$$

Avec :

$$i = (l-1) \times p + u$$
$$j = (m-1) \times p + v$$

L'équation (3.1) nous permet d'établir la relation entre les composants de la matrice des fonctions BSplines B et la matrice des fonctions de bases realative à l'intervalle modifié comme suit :
Soit k l'indice de l'intervalle modifié, on a :

$$B = A \otimes B_k \otimes C$$

Avec :

- $A = B_1 \otimes B_2 \otimes ... \otimes B_{k-1}$
- $C = B_{k+1} \otimes B_{k+2} \otimes ... \otimes B_n$
- $B \in M_N(\mathbf{R})$ où $N = \prod_{k=1}^{p} \beta_k$ et $\forall k \in [\![1, p]\!], \beta_k$ est l'ordre de la variable q_k dans le polynôme F

On pose $B = (B_{i,j})_{1 \leq i,j \leq N}$ et $B_k = (b_{i,j})_{1 \leq i,j \leq \beta_k \times}$, on a :

$$\forall i,j \in [\![1,N]\!], \exists u,v \in [\![1,\beta_k]\!] \text{ tels que : } B_{i,j} = b_{u,v} \times K \text{ (avec } K \in \mathbf{N}) \tag{3.2}$$

Où :

$$u = [i\%(N_1 \times \beta_k)] \div N_1 + 1$$
$$v = [j\%(N_1 \times \beta_k)] \div N_1 + 1$$

Avec :

- $N_1 = \prod_{i=k+1}^{p} \beta_i$
- % est l'opérateur modulo qui retourne le reste de la division euclidienne
- $\div$ est l'opérateur de la division entière qui retourne le quotient de la division euclidienne

3.3.2 Pseudo-code

En se basant sur les relations établies dans la partie précédente et l'équation (3.2), on peut écrire le pseudo-code du programme permettant la mise à jour de la matrice B à partir de la matrice B_k.

Algorithm 2: Mise à jour de la matrice B

Entrée: k l'indice de l'intervalle modifié ,(β_i) l'ordre de chaque variable dans la fonction F, L'ancienne valeur de la matrice B $Basis$, L'ancienne valeur de la matrice B_k $prev_B_k$, p la dimension de la fonction F

Sortie : La matrice $Basis$ mise à jour

$N \leftarrow \prod_{i=1}^{p}(\beta_i + 1)$
$N_1 \leftarrow \prod_{i=1}^{p}(k - 1)$
$new_B_k \leftarrow$ La matrice des fonctions de base caculée avec la nouvelle valeur de q_k
for $1 \leq i \leq N$ **do**
  for $1 \leq j \leq N$ **do**
    $u \leftarrow [i\%(N_1 \times \beta_k)] \div N_1 + 1$
    $v \leftarrow s[j\%(N_1 \times \beta_k)] \div N_1 + 1$
    $Basis[i][j] \leftarrow \frac{Basis_{i,j} \times new_B_k[u][v]}{prev_B_k[u][v]}$
  end for
end for
return $Basis$

En analysant la complexité temporelle de l'algorithme ci-dessus, on conclut que :

- L'initialisation des variables N et N_1 se fait en $O(n)$
- Le calcul de la nouvelle matrice B_k se fait en $O(n^2)$
- Les trois affectations qui s'exécutent à l'intérieur des boucles se font en $O(1)$ donc les boucles ont une complexité temporelle en $O((N_{max})^{2+p})$ avec $\beta_{max} = \max_i \beta_i$

On peut donc déduire que la complexité temporelle de l'algorithme de mise à jour de la matrice B est en $O(e^{n \times log(n)})$.

3.4 Normalisation et mise à jour des coefficients du polynôme

3.4.1 Principe de la méthode

Cette méthode consiste à éliminer la dépendance entre la matrices des fonctions de base et les bornes des intervalles passés en entrée à la fonction F. Les calculs seront donc faits de façon à ce que la matrice B soit constante et le vecteur X varie en fonciton des bornes des intervalles. Pour celà, on effectue une normalisation de données comme suit : $\forall i \in [\![1, p]\!]$:

$$[q_i] = m_i + \frac{d_i}{2} \times [q_i]^{ref}$$

où m_i et d_i sont le centre et le diamètre de l'intervalle $[q_i]$ respectivement. Et on initialise les intervalles de référence $[q_i]^{ref}$ par $[-1, 1]$ pour tout i dans $[\![1, p]\!]$ Ainsi, la matrice B ne dépend que des intervalles de de référence qui eux restent constants, donc B est constante dans ce cas. Étant donné que seules les fonctions polynômiales sont considérées dans ce projet, on pose :

$$F([q]) = \sum_{i=0}^{n} \alpha_i \times \prod_{k=1}^{p} [q_i]^{\beta_{i,k}}$$

avec :

- n est le nombre de monômes de la fonction F
- p est la dimension de la fonciton F
- $[q] = [q_1] \times [q_2] \times ... \times [q_p]$ est un vecteur d'intervalles
- $\{\alpha_1, \alpha_2, ..., \alpha_n\}$ est l'ensemble des coefficients du polynôme F
- $(\beta_{i,j})_{1 \leq j \leq n}$ est l'ensemble des degrés de la variable q_i pour tout i dans $[\![1, p]\!]$

En effectuant la normalisation des intervalles, on a :

$$
\begin{aligned}
F([q]) &= \sum_{i=0}^{n} \alpha_i \times \prod_{k=1}^{p} (m_i + \frac{d_i}{2} \times [q_i]^{ref})^{\beta_{i,k}} \\
&= \sum_{i=0}^{n} \alpha_i \times \prod_{j=1}^{p} \sum_{k=0}^{\beta_{i,j}} \binom{\beta_{i,k}}{k} m_i^{\beta_{i,k}-k} \times (\frac{d_i}{2} \times [q_i]^{ref})^k \\
&= \sum_{i=0}^{n} \prod_{j=1}^{p} \frac{\alpha_i \times (m_i^{\beta_{i,j}})}{\beta_{i,j}!} \sum_{k=0}^{\beta_{i,j}} \frac{1}{(k+1) \times (k+2) \times ... \times (\beta_{i,j} - j)} \times (\frac{d_i}{2 \times m_i} \times [q_i]^{ref})^k \quad --(x)
\end{aligned}
$$

L'équation (x) nous permet de calculer les éléments du vecteur X qui représentent les coefficients de la fonction F et les bornes des intervalles.

3.4.2 Pseudo-code

L'equation (x) nous permet d'établir l'agorithme de mise à jour du vecteur X :

Algorithm 3: Mise à jour du vecteur X

```
Entrée: α_i ,(β_{i,j}), m_i, d_i, p
Sortie : Le vecteur X mis à jour
Initialiser X par le polynôme nul
for 1 ≤ i ≤ n do
    Y ← le polynôme nul
    for 1 ≤ j ≤ p do
        c_{i,j} ← (α_i × m_j^{β_{i,j}}) / β_{i,j}!
        Y' ← le polynôme nul
        for 1 ≤ k ≤ β_{i,j} do
            C_{k,j} ← 1/((k+1)×(k+2)×...×(β_{i,j}−j)) × (d_i/(2×m_i) ×)^k
            //Y'_k désigne le coefficient du k^ème monôme du polynôme Y'
            Y'_k ← c × C_{k,j}
        end for
        Y ← Y × Y'
    end for
    X = X + Y
end for
return X
```

On évalue la complexité temporelle de cet algorithme en analysant chaque boucle constituant le programme.

- La boucle sur les $\beta_{i,j}$ effectue à chaque itération des calcul en $O(n)$, elle est donc en $O(n^2)$
- La boucle sur les j effectue à chaque itération deux affectations en $O(n)$ en plus de la boucle analysée précédemment, elle a donc une complexité en $O()$
- La boucle extérieure fait N_{max} itérations en $O()$ chacune, avec N_{max} est le nombre total de monômes dans la fonction F, donc $N_{max} \rightarrow (\beta_{max})^p$ où $\beta_{max} = \max_{i,j} \beta_{i,j}$. La conplexité temporelle de cette boucle est donc en $O()$

L'analyse effectuée ci-dessus nous permet de conclure la complexité de l'algorithme de mise à jour du vecteur X : Il s'agit d'une complexité exponentielle en $O(e^{n \times log(n)})$

3.5 Comparaison des méthodes et Conclusion

Après l'analyse de la complexité temporelle des deux méthodes proposées, il s'est avéré que les deux méthodes ont une complexité exponentielle en $O(e^{n \times log(n)})$.
Dans la première méthode les valeurs des matrices B et B_k calculées à chaque itéraion sont réutilisées dans l'itération suivante. Ceci induit l'augmentation des troncatures appliquées sur ces valeurs à chaque itération d'où la multiplication des erreurs d'arrondis au fur et à mesure de l'exécution du programme.
Ce problème ne se pose pas au niveau de la deuxième méthode, qui - elle - consiste à recalculer les coeffcients du vecteur X à zéro, ce qui réduit les taux d'érreurs dûes aux arrondis des nombres réels.
On peut déduire alors que cette dernière méthode présente plus de précision au niveau des résultats avec un temps d'éxécution très similaire à celui de la première méthode.
Nous avons donc choisi d'implémenter la deuxième méthode pour garantir de meilleurs résultats et un temps d'éxécution optimal.
Les résultats obtenus seront mis en oeuvre dans la partie suivante.

Chapitre 4

Mise en oeuvre et résultats

Dans cette dernière partie nous allons présenter les résultats obtenus après l'implémentation des méthodes de résolution des POC étudiées dans différents langages de programmation ainsi que la planification des tâches réalisées tout au long de ce travail.

4.1 Résultats et Interprétations

4.1.1 Langages de programmations utilisés

Les langages de programmation utilisées dans notre projet sont C++ et Python afin de comparer les performances des deux en termes de temps de calcul.

4.1.2 Mise en oeuvre des résultats

Pout les tests, on considère la fonction critère $f(a,b) = (a+b)^2$. On utilisera les deux formulations factorisée et développé suivantes en plus des fonctions BSplines, pour mettre en avant le problème de pessimisme dû au nombre d'occurences des variables :

$$f_1(a,b) = (a+b)^2$$
$$f_2(a,b) = a^2 + b^2 + 2ab$$

	a	b
Intervalles:	[-1,1]	[3,4]
fonction critère	min (a+b)²	
Contrantes 1	x + y >= 0	
Contrantes 2	x² + y -2 > 0	
Contrantes 3	3x + y > 2	

Figure 4.1 – Résumé des données du problème

Le tableau ci-dessus présente les données du problème sur lequel les tests on été effectués.

		Exec Tmps	Nb iterations	Min Value	Boîte
Formule 1 (x + y)^2	python	3.1943016052246094	38877	7.123269081115723	[-1 , -0.9990234375] x [3 , 3.0009765625]
	c++	2.513	38877	7.12327	[-0.333008,-0.332031] and [3,3.00098]
Formule 2 x^2 + y^2 + 2xy	python	4.875219106674194	40985	7.124567031860352	[-1 , -0.9990234375] x [3 , 3.0009765625]
	c++	128.29	40985	7.12457	[-0.333008,-0.332031] and [3,3.00098]
Bsplines	python	2.875219106674194	3930	7.124567031860352	[-1 , -0.9990234375] x [3 , 3.0009765625]
	c++	1.29	3930	7.12457	[-0.333008,-0.332031] and [3,3.00098]

FIGURE 4.2 – Résultats de la bissection

En analysant les résultats, on constate que le nombre d'itération est plus important avec la formule développée car son expression accentue le pessimisme. On remarque également la contribution considérable des BSplines dans la réduction du nombre d'itérations et du temps d'exécution à travers la réduction du pessimisme.

4.2 Gestion du projet

4.2.1 Organisation de l'équipe du projet

Le projet est à réaliser en binome, donc pour bien gérer notre temps et pouvoir cerner le travail demandé, nous avons décidé que chacune de nous développera les algorithmes avac un language de programmation précis, autrement dit, nous avons travaillé sur la conception tous les deux, et l'implémentation séparément.

4.2.2 Intégration continue

Pour pouvoir gérer les différentes versions tout au long de ce projet, nous avons utilisé la plateforme Forge qui dispose d'un repertoire git ce qui nous a faciliter la collaboration à distance et assurer l'intération continue de notre code ainsi que la documentation de notre travail.

4.2.3 Réunions hebdomadaires

Le regroupement été organisé sous forme des réunion hebdomadaires qui nous ont permis de travailler constamment sur notre projet et d'avancer plus rapidement et efficacement sur le sujet. Dans chaque réunion, on discute les problèmes rencontrés pendant la semaine, on essaye de corrige les erreurs commis et on précise les taches à faire pour la prochaine réunion.

4.2.4 Déroulement du travail

Le projet a commencé le 20 novembre et nous avions jusqu'à le 1 mars 2021 pour le finir.
Ce travail a été réalisé selon 3 phases principales :

- **Comprehension de projet :** cette phase consiste à bien saisir les notions de notre projet en se basant principalement sur la thèse de Rawan KALAWOUN ainsi que des articles scientifiques et des cours disponible en ligne.

- **Bissection :** dans cette phase, nous avons implémenté l'algorithme de bissection en deux étapes, la première sans contraintes pour bien comprendre le fonctionnement de l'algorithme et la deuxième en ajoutant une fonction qui evalue les différentes contraintes. Et pour tester, nous avons exécuté l'algorithme avec des différentes formulations pour la fonction critère ainsi que les fonctions contraintes.
- **BSplines :** Dans cette partie, nous avons implémenté les BSplines en utilisant la formule de Cox-de Boor et nous les avons intégré dans l'algorithme de bissection.

4.2.5 Diagramme de Gantt

Ci-dessous le diagramme de Gantt modélisant le déroulement de notre travail :

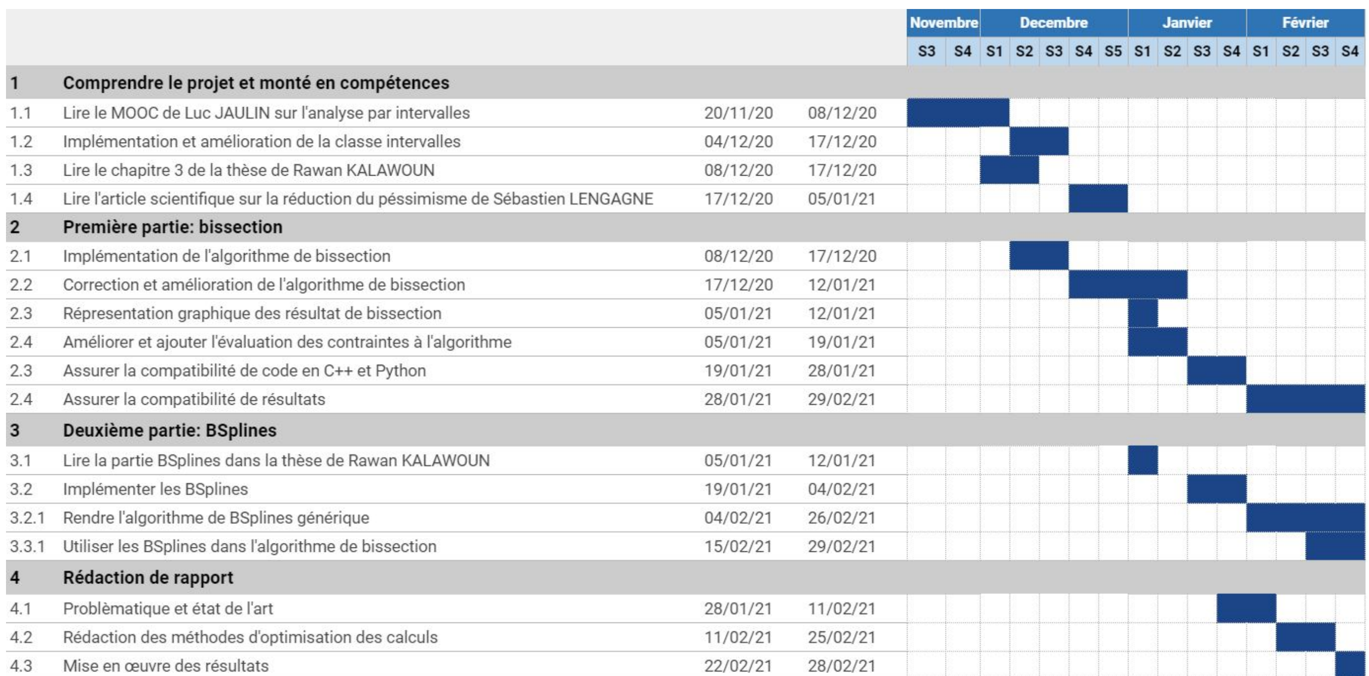

Figure 4.3 – Diagramme de Gantt

Conclusion

En conclusion, notre travail tout au long de ce projet a porté sur deux parties importantes :

- La première étant celle de l'étude approfondie de l'état de l'art basé principalement sur la thèse de Rawan Kalawoun
- La deuxième partie concernait notre notre valeur ajoutée au projet et consistait à analyser et implémenter les deux méthodes proposées pour l'amélioration de l'algorithme d'optimisation.

Ce projet a été assez particulier, notamment parce que le sujet est basé sur une thèse de doctorat et nous a demandé un effort considérable pour assimiler toutes les notions dont on avait besoin au niveau l'implémentation de l'algorithme. Ceci nous a permis de monter en compétences, non seulement sur le plan technique, mais aussi au niveau des connaissances mathématiques. Ce projet nous a aussi donné également l'occasion de frôler le monde de la recherche qui nécessite autant de passion que de patience.

Annexe A

La bissection

Pour comprendre l'algorithme de bissection, prenant l'exemple suivant :
Fonction critère :

$$f(x, y) = (x + y)^2$$

Les contraintes :

$$x^2 - 1 \geq 0$$

$$x + y - 5 \leq 1$$

$$x + y^2 \geq 2$$

Avec une précision $\epsilon = 4$.
Le but est de trouver la valeur minimale qu'on peut obtenir.
La figure ci-dessous représente le résultat de l'algorithme de bissection avec $x = [-4, 3]$ et $y = [-2, 5]$.
L'algorithme consiste a diviser une dimention de la boite a chaque itération a condition que la précision est supérieur à un seuil donné (ϵ), autrement dit, le diamétre de l'intervalle à diviser soit supérieur au seuil. Si tous les dimensions peuvent etre bissecter, celui avec le plus grand diamétre est choisi, et si ils ont le meme diamétre, le choix est arbitraire.
A chaque itération, et avant décider selon quelle dimention la division sera faite, une vérification de la satisfaction des contraintes est nécessaire et on distingue trois cas :

- La contrainte est totalement validée donc on passe à la contrainte suivante. Si les contraintes sont toutes satisfaites on vérifie si cette boite donne un résultat mieux que la boite optimume, si oui elle devient l'optimume sinon on passe à une autre.
- La contrainte est partiellement validée, on bissecte la boite.
- Si La contrainte n'est pas respectée, la boite est rejetée.

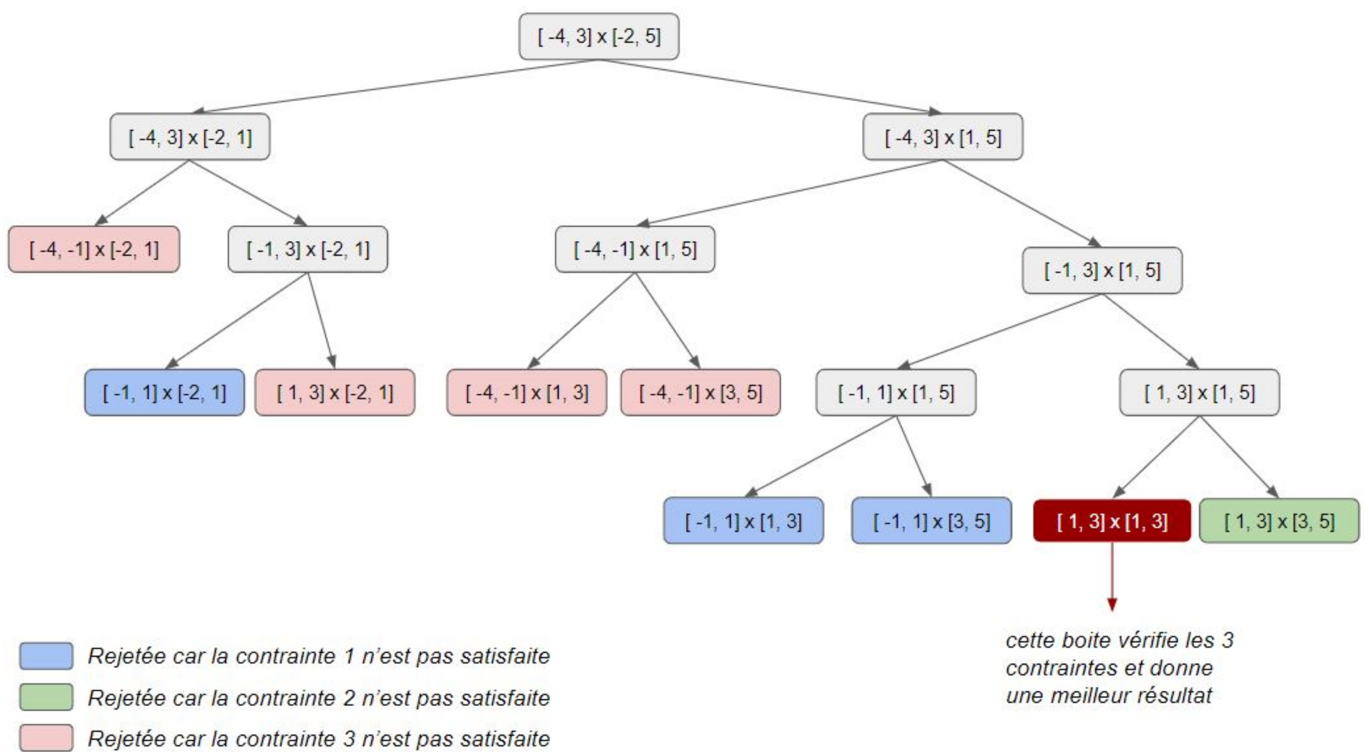

Figure A.1 – Application de l'algorithme de bissection sur 2 dimensions

Bibliographie

[1] Rawan Kalawoun, *Motion planning of multi-robot system for airplane stripping.*

[2] Wikipedia, *De Boor's Algorthm.*

[3] Geeks for Geeks, *Dynamic Programming.*

[4] Wikipédia, *Produit de Kronecker.*

[5] Luc Jaulin, *Analyse par intervalles*