

# Initiation à la Recherche

## *Conception et réalisation d'un logiciel interactif de visualisation de pavages 2D/3D*

A.MARGUERITE  
R.RINCÉ

Université de Nantes  
2 rue de la Houssinière, BP92208, F-44322 Nantes cedex 03, FRANCE



# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Présentation du sujet . . . . .	3
1.2	Équipe d'accueil . . . . .	3
1.3	Déroulement du projet . . . . .	3
1.4	Caractéristiques de l'outil . . . . .	4
<b>2</b>	<b>Domaine de recherche</b>	<b>5</b>
2.1	Introduction . . . . .	5
2.2	Présentation du problème . . . . .	5
2.2.1	Définition du contexte . . . . .	5
2.2.2	Exemple . . . . .	5
2.3	Méthodes de calculs . . . . .	6
2.3.1	Méthodes formelles . . . . .	6
2.3.2	Méthodes numériques . . . . .	6
2.4	Résolution par Intervalles . . . . .	6
2.4.1	L'Arithmétique des intervalles . . . . .	7
2.4.2	Utilisation des intervalles pour la notion de contraintes . . . . .	7
2.4.3	Exemples d'application . . . . .	8
2.5	Mise en œuvre de la méthode de résolution par intervalles . . . . .	10
2.5.1	Algorithmes . . . . .	10
2.5.2	Données en sortie . . . . .	10
2.6	Applications par des outils . . . . .	11
2.6.1	Realpaver . . . . .	11
2.6.2	Autres outils existants . . . . .	13
<b>3</b>	<b>Contributions</b>	<b>14</b>
3.1	Introduction . . . . .	14
3.2	Boîtes et pavage . . . . .	14
3.2.1	Boîte . . . . .	14
3.2.2	Pavage . . . . .	16
3.3	Stockage des boîtes et visualisation . . . . .	18
3.3.1	Introduction . . . . .	18
3.3.2	Étude d'une solution possible : le QuadTree . . . . .	18

3.3.3	Étude d'une seconde solution : le R-tree . . . . .	21
3.3.4	Conclusion . . . . .	24
<b>4</b>	<b>Étude pratique</b>	<b>25</b>
4.1	Chargement de pavages . . . . .	25
4.1.1	Map internes . . . . .	25
4.1.2	Listes et Tableaux internes . . . . .	26
4.1.3	Maps globales . . . . .	26
4.2	Accès au élément d'une boîte . . . . .	27
4.2.1	Tests d'accès aux caractéristiques . . . . .	27
4.2.2	Tests d'accès aux coordonnées . . . . .	28
4.3	Conclusion . . . . .	28
4.4	Annexes . . . . .	29
4.4.1	Études de performance de création d'une boîte . . . . .	29
4.4.2	Études de performance d'accès aux éléments d'une boîte . . . . .	33
<b>5</b>	<b>Conclusion</b>	<b>36</b>
<b>A</b>	<b>Annexes</b>	<b>37</b>
A.1	Cahier des charges . . . . .	37
A.2	Documents de spécifications . . . . .	40

# 1 Introduction

## 1.1 Présentation du sujet

Le module *Initiation à la recherche* du 2<sup>nd</sup> semestre du master ALMA propose une initiation au métier de chercheur en informatique. Dans le cadre de ce module notre binôme a choisi le sujet : *Conception et réalisation d'un logiciel interactif de visualisation de pavages 2D/3D*. Ce sujet à la particularité d'être la reprise d'un projet de stage. Ce stage effectué au cours de l'été 2011 par un des membres du binôme avait pour objectif initial la conception et la réalisation d'un outil de visualisation des calculs en sortie du logiciel *Realpaver* (cf. Section 2.6.1) développé au sein de l'équipe d'accueil. Le développement de ce stage a suivi le modèle du cycle en V.

Au terme des deux mois de stage, le travail accompli fut le suivant :

1. Rédaction du cahier des charges (cf. Annexe A.1)
2. Rédaction du document de spécification (cf. Annexe A.2).

## 1.2 Équipe d'accueil

L'équipe Optimisation globale, optimisation multi-objectifs[[opt](#)] (OPTI) du laboratoire LINA[[lin](#)] travaille principalement sur des méthodes visant à la résolution efficace de problèmes d'optimisation complexes.

## 1.3 Déroulement du projet

Au cours des premières semaines, C.JERMANN nous a proposé d'étudier de manières générales les sujets de l'équipe OPTI. Un bref bilan de cette étude est proposé dans le chapitre 2. En parallèle, nous avons manipulé quelques outils ad-hoc tel que *gnuplot* [[gnu](#)], utilisés jusqu'à présent par l'équipe OPTI pour représenter les valuations calculées par *Realpaver*. Par la suite nous avons pris (ou repris) connaissance du cahier des charges et du document de spécifications, en apportant certaines corrections à ce dernier.

Une problématique est apparue quant à l'axe d'étude à choisir pour le déroulement du projet. En effet une des caractéristiques principale de l'outil à concevoir, est de manipuler

un nombre potentiellement très grand de données (*cf.* Chapitre 3). Nos alternatives étaient alors les suivantes :

1. Étudier les différents algorithmes et structures de données nécessaires à l'outil.
2. Entamer directement la conception de l'outil (choix de design pattern, IHM, ...).

Ce second choix impliquait la possibilité au terme de l'année, de proposer un résultat « matériel ». En effet le document de spécifications étant très concis, nous aurions pu très rapidement produire du code en le suivant à la lettre. Cependant cette démarche ne répondait pas aux objectifs de découverte du module d'Initiation à la Recherche. De plus, passer outre cette étude de structures et d'algorithmes, aurait rapidement entraîné la conception dans une impasse. Nous avons donc choisi la seconde alternative. Notre contribution à cette étude est résumé dans le chapitre 3 de ce document.

## 1.4 Caractéristiques de l'outil

Nous rappellerons dans cette section quelques notions propres à l'outils et indispensables à la compréhension de cette étude.

**Données en entrée** Un format spécifique à l'outil est défini dans le document de spécification(*cf.* Annexe A.2).

**Fenêtre de visualisation** Il s'agit de la partie du pavage visible à l'écran à un instant  $t$ . Selon le cahier des charges, l'utilisateur peut « naviguer » dans l'environnement contenant le pavage. La gestion de la fenêtre de visualisation est l'une des problématiques abordées dans cette étude (*cf.* Section 3.3).

**Dimensions visualisées** L'intitulé du sujet d'étude précise que l'outil doit permettre une visualisation de deux ou trois dimensions. Cependant *Realpaver* est en mesure de calculer des problèmes à  $n$  dimensions. L'utilisateur doit donc préciser les deux ou trois variables sur lesquelles seront effectuées la projection du pavage.

**Filtre** Le cahier des charges demande la possibilité d'appliquer des filtres sur le pavage. Il s'agit d'effectuer une classification des boîtes déterminée par l'utilisateur (*cf.* Annexe A.2 section 1.2, 5<sup>ème</sup> point). Une telle opération sur le pavage va nécessiter de nombreuses opérations (accès aux caractéristiques d'une boîte, tris, ...). La mise en œuvre de ces opération est l'une des problématiques principale de la conception.

**Données en sortie** Une sauvegarde du pavage visualisé est demandé par le cahier des charges. Il s'agit pour l'outil d'une simple « sauvegarde » utilisant le même format que le fichier d'entrée. La création d'images à partir de l'écran de visualisation est aussi demandée.(*cf.* Annexe A.2)

## 2 Domaine de recherche

### 2.1 Introduction

L'objectif de ce document est de décrire les notions essentielles à retenir en ce début de projet d'initiation à la recherche. Ces notions font partie d'un même sujet d'étude, au cœur des travaux de l'équipe OPTI. Elles concernent le sujet des contraintes et des intervalles.

### 2.2 Présentation du problème

#### 2.2.1 Définition du contexte

Un problème de satisfaction de contraintes (CSP) est défini par 3 éléments :

- Un ensemble de variables  $\mathbf{V} = \{v_1, \dots, v_n\}$ .
- Un domaine de valeurs pour chaque variable. Chaque valeur du domaine  $D_i$  associé à la variable  $v_i$  est une valeur que peut potentiellement prendre  $v_i$  :  $\mathbf{D} = D_1 \times \dots \times D_n$ .
- Un ensemble de contraintes (relations)  $\mathbf{C}$  restreignant les variables de  $\mathbf{V}$  défini ci-dessus :  $\mathbf{C} = \{c_1, \dots, c_m\}$ .

On s'intéressera notamment à la résolution des CSP sur le domaine continu. Dans ce cas les domaines des variables seront des intervalles dans les réels et les contraintes seront généralement représentées par des équations et des inégalités.

#### 2.2.2 Exemple

Une représentation simple de ce type de problèmes est la recherche des intersections de deux cercles dans un plan. Définissons deux cercles respectivement de rayons  $r_1$  et  $r_2$  et de centres :  $(x_1, y_1)$  et  $(x_2, y_2)$ . On peut alors fixer les données en entrées du problème :

- Un ensemble de variables  $\{x, y\}$ .
- Un domaine de valeurs pour chaque variable.
- Les constantes du problème : le rayon de chaque cercle situé dans  $\mathbb{R}^+$ .
- L'ensemble des contraintes est composé par les équations des cercles :

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 = r_1^2 \\ (x - x_2)^2 + (y - y_2)^2 = r_2^2 \end{cases} \quad (2.1)$$

## 2.3 Méthodes de calculs

Les méthodes de calculs permettent la résolution de problèmes mathématiques en machine ; en particulier lorsque l'on cherche à résoudre des problèmes sur les nombres réels. Elles permettent par exemple la résolution des CSP ou GCSP (Geometric Constraint Satisfaction Problem [[C.JERMANN02](#)]) et peuvent être divisées en deux catégories : les méthodes formelles et les méthodes numériques.

### 2.3.1 Méthodes formelles

Les méthodes formelles permettent de résoudre des systèmes d'équations ou d'inéquations en utilisant au maximum le calcul symbolique. Les approches les plus classiques des méthodes formelles utilisent des théories, telles que les idéaux polynomiaux pour les bases de Gröbner, ou la théorie des déterminants pour la méthode du résultant.

Ces méthodes ont l'énorme avantage de retourner des solutions exactes d'un système d'équations, ou tout du moins de minimiser l'utilisation de l'arithmétique flottante. Elles tenteront donc de résoudre le problème [2.1](#) en effectuant uniquement des opérations sur les équations sans évaluer les variables. On pourra donc avoir des expressions pour  $x$  et  $y$  représentant les solutions du problème de façon symbolique.

Cependant les résolutions de problèmes par des méthodes formelles sont forcément restreintes par les possibilités du calcul symbolique et ne pourront donc pas toujours offrir de solution générale et met en œuvre des algorithmes de complexité exponentielles. Il n'existe par exemple pas de formules générales permettant de trouver les solutions d'un polynôme de degré supérieur ou égal à cinq.

### 2.3.2 Méthodes numériques

Les méthodes numériques consistent à évaluer de façon calculatoire la ou les solutions d'un problème. En effet elles sont capables d'essayer de résoudre n'importe quel système d'équations (ou d'égalités). Ainsi dans le cas du problème [2.1](#) la machine va effectuer directement les calculs pour évaluer le résultats. Or l'utilisation de la représentation flottante et de son arithmétique pour simuler les opérations réelles va entraîner une diffusion et une augmentation de l'erreur de calcul, à tel point que l'on ne peut parfois plus assurer la validité d'une solution. On pourra d'ailleurs citer à titre d'exemple le problème de l'inversion d'une matrice mal conditionnée[[Wikb](#)].

Cependant ces calculs numériques utilisés par des méthodes de résolutions par intervalles permettent de contourner ces problèmes. Pour plus de détails sur la représentation flottante, on pourra se référer à la thèse de Frédéric GOUALARD<sup>1</sup> [[F.GOUALARD00](#)].

## 2.4 Résolution par Intervalles

Les méthodes formelles et numériques, bien que performantes par certains aspects, sont rapidement limitées lorsque l'on veut résoudre des problèmes complexes ou que

---

1. Notamment dans la première Partie : *Chapitre 1 : L'arithmétique flottante*

l'on veut pouvoir valider une solution (problème de propagation des erreurs...). C'est dans ce cadre que la méthode de résolution par intervalles a toute sa place. Construite grâce à l'arithmétique des intervalles, elle utilise aussi des notions apportées par la programmation par contraintes.

### 2.4.1 L'Arithmétique des intervalles

Cette arithmétique permet un calcul sur l'ensemble  $\mathbb{I}$  des intervalles à bornes sur  $\mathbb{R}$ . Les bornes  $b_1$  et  $b_2$  de l'intervalle  $[b_1, b_2]$ , résultant de tout calcul, sont choisies en prenant un arrondi respectivement inférieur à  $b_1$  et supérieur à  $b_2$  de manière à garantir l'exactitude des calculs. L'extension des fonctions aux intervalles, introduite par MOORE en 1966, permet une transition à des intervalles grâce à un opérateur d'encadrement. Les opérations élémentaires sont alors réécrites en terme des bornes des intervalles, par exemple :

$$\begin{cases} [a..b] + [c..d] = [a + c..b + d] \\ [a..b] \times [c..d] = [\min(ac, ad, bc, bd)..\max(ac, ad, bc, bd)] \end{cases} \quad (2.2)$$

Dans le cas d'une fonction non monotone comme le cosinus, on découpe les intervalles en domaines où la fonction est monotone (2.1)

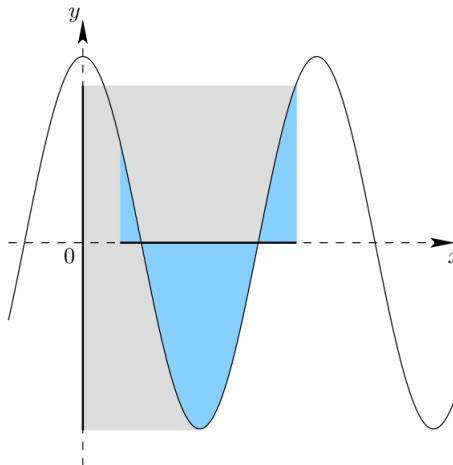


FIGURE 2.1 – Cas de la fonction cosinus

Une liste non-exhaustive des opérations de cet opérateur est listée dans [F.GOUALARD00].

### 2.4.2 Utilisation des intervalles pour la notion de contraintes

Également, la méthode de résolution par intervalles utilise des notions de programmation par contraintes. On y retrouve celle de consistance. La consistance consiste à rechercher les valeurs cohérentes dans le domaine des variables pour les contraintes du CSP. Par exemple un CSP est globalement consistant lorsque toutes les valeurs des



variables de son domaine appartiennent au moins à une solution. On devine qu'il peut être intéressant pour un CSP de posséder la consistance la plus forte possible. Ainsi les opérations pour la résolution de problèmes seront moins nombreuses.

On visualise ici que les solutions sont encadrées par des « boîtes » rouges, et non réduites à des points.

Dans le cas du problème 2.2, il serait possible d'avoir des solutions différentes selon la consistance choisie. Avec une hull-consistance, nous obtiendrions une boîte contenant les deux solutions mais aussi une bonne partie de l'intersection des deux cercles. La *3B-consistance* est une méthode plus puissante. Cette méthode vérifie, à l'instanciation de deux variables, vérifie localement toutes les contraintes. Une illustration des différences entre ces méthodes est proposée sur la figure 2.2. Le problème résolu est celui de l'intersection de deux disques.

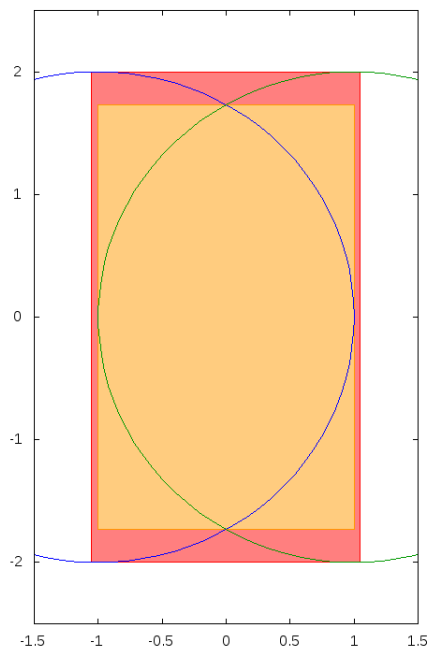


FIGURE 2.2 – Comparaison graphique entre la Hull-consistance (rouge) et la 3B-consistance (orange)

### 2.4.3 Exemples d'application

La précision de la méthode attirent le monde industriel. Notamment les applications composées de contraintes géométriques comme la robotique. La figure 2.3 illustre un robot avec deux points d'encrage ( $P_1$  et  $P_2$ ) possédant chacun un bras rotatif de longueurs

respectives  $[L_1]$  et  $[L_2]$  et orientés respectivement par les angles  $\alpha_1$  et  $\alpha_2$ . Ces bras sont composés de pistons permettant d'ajuster leurs longueurs. Ainsi leurs longueurs de bras sont comprises respectivement entre  $[L_{1min} \dots L_{1max}]$  et  $[L_{2min} \dots L_{2max}]$  et les angles entre  $[\alpha_{1min} \dots \alpha_{1max}]$  et  $[\alpha_{2min} \dots \alpha_{2max}]$ .

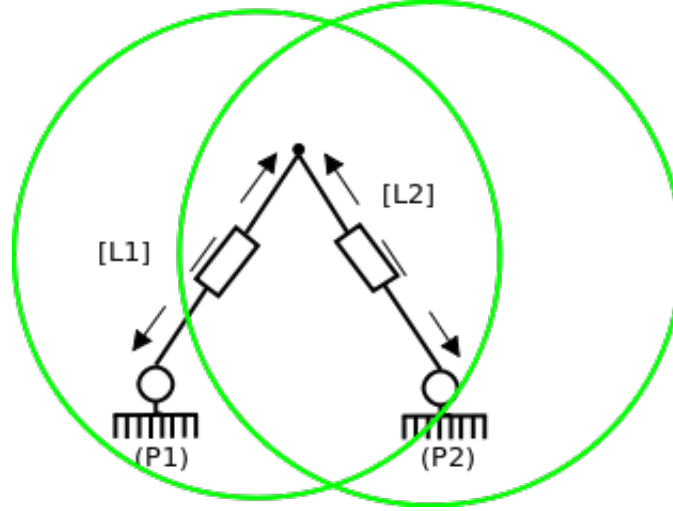


FIGURE 2.3 – Exemples d'application dans la robotique

L'objectif est de connaître l'ensemble des emplacements possibles pour la main du robots en fonction des paramètres donnés ci-dessus. Dans le cas de la figure 2.3, on peut considérer que les  $L_{imin}$  sont nulles et que les bras puissent effectuer des rotations de  $360^\circ$ . Ainsi le problème consiste à retrouver l'intersection de deux disques.

Par ailleurs, ces méthodes intéressent régulièrement les industriels. On retrouve en effet dans [H.SCHICHL03], les détails d'applications dans le secteur de la chimie industrielle mais aussi dans celui de la biologie avec une étude sur les protéines. La science fondamentale met aussi en application ces outils. C'est le cas en mathématique par exemple de problèmes tel que la conjecture de KEPLER ou encore le maximum de clique.

## 2.5 Mise en œuvre de la méthode de résolution par intervalles

### 2.5.1 Algorithmes

L'algorithme de *Branch and Prune* est au centre de la méthode de résolution par intervalles. Nous résumons ici ses deux étapes :

**Branch :**

*Diviser pour mieux régner*

Cette méthode consiste à diviser récursivement un problème en deux sous problèmes. Appliquée à la méthode de résolution par intervalles, on découpe en deux l'intervalle concerné (en son milieu ou non). On obtient alors deux problèmes plus « petits » à étudier.

**Prune :** La découpe d'un problème en sous problèmes peut amener à une situation où un des sous problèmes créés ne contient aucune solution. On peut alors étudier et trier ces sous problèmes pour en supprimer les espaces de recherche superflus. Cette étape est appelée *pruning*.

Au bout d'un certain nombre d'itération, l'algorithme de *Branch and Prune* propose un ensemble de solutions sous la forme d'intervalles.

### 2.5.2 Données en sortie

En sortie l'algorithme va nous retourner un ensemble de boîtes. Cet ensemble garantissant, à contrario des méthodes numériques classiques, que toutes les solutions y sont incluses. Dans la section 2.6.1 un exemple d'illustration d'un tel ensemble est donné.

Il est d'ailleurs possible de savoir si tout l'espace d'une boîte est solution du problème. En contrepartie on ne peut toujours garantir l'existence d'une solution dans une enveloppe, il peut donc exister une approximation autour des résultats de l'algorithme.

## 2.6 Applications par des outils

### 2.6.1 Realpaver

Développé au sein de l'équipe OPTI, cet outil reprend, entre autres, les méthodes de résolutions de contraintes géométriques par propagation d'intervalles. Il permet la résolution de systèmes d'équations à  $n$  variables représentant des contraintes. Dans un soucis de précision ces solutions sont sous la forme d'intervalles [LF06].

**Problème** Soient deux disques, l'un de centre  $(-1, 0)$  et l'autre de centre  $(1, 0)$ . Tous deux sont de rayon 2. Donnez l'ensemble des solutions de l'intersection de ces deux disques.

**Modèle** L'outil *Realpaver* modélise un tel problème de la sorte :

```
Constants
  x0 = -1,
  y0 = 0,
  R0 = 2,
  x1 = 1,
  y1 = 0,
  R1 = 2
;
Variables
  x in ]-oo, +oo[,
  y in ]-oo, +oo[
;
Constraints
  (x-x0)^2 + (y-y0)^2 <= R0^2,
  (x-x1)^2 + (y-y1)^2 <= R1^2
;
```

**Données en sortie** L'ensemble de solutions suivant *Realpaver* propose l'ensemble de solutions suivant :

RealPaver v. 0.4 (c) LINA 2004

```
INITIAL BOX
  x in ]-oo , +oo[
  y in ]-oo , +oo[

OUTER BOX 1
  x in [0.7924334198270921 , 0.795689483022687]
  y in [0.8806243697296342 , 0.8872330220900007]
```

...

OUTER BOX 3053

x in [-0.7956894830226868 , -0.7924334198270919]

y in [-0.8872330220900011 , -0.8806243697296345]

precision: 0.00661, elapsed time: 150 ms

END OF SOLVING

Property: reliable process (no solution is lost)

Elapsed time: 150 ms

Ces 3053 solutions proposées par *Realpaver* sont illustrées sur la figure 2.4.

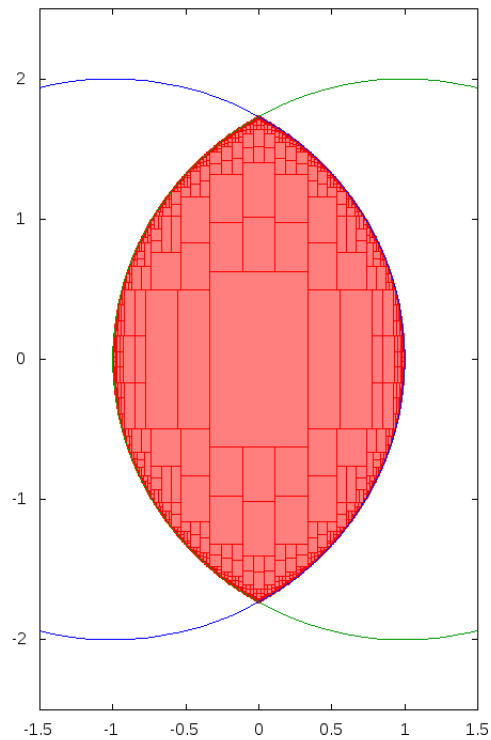


FIGURE 2.4 – Intersection de deux disques

### 2.6.2 Autres outils existants

**CHOCO** Projet commun de l'École des Mines de Nantes, de LIRMM de Montpellier ainsi que l'INRA de Toulouse. CHOCO est une librairie JAVA dédiée au résolution de CSP et à la programmation par contrainte. [[cho99](#)]



**Google or-tools** Bibliothèque Python fournissant un solveur de contraintes générique et divers algorithmes, notamment sur les problèmes de graphes ou de sacs à dos : [[ort11](#)]

## 3 Contributions

### 3.1 Introduction

L'objectif de ce chapitre est de mener une étude sur les différentes structures de données nécessaires aux futurs algorithmes de visualisation. Dans une première partie, nous nous intéresserons à l'opération d'accès à une caractéristique ou d'une donnée d'une boîte ainsi qu'à l'occupation mémoire de cette dernière. Les problématiques liées à l'implémentation d'un pavage (ensemble de boîtes) est par la suite abordé. Il s'appuie sur le document de spécification (*cf.* Annexe [A.2](#)). Ce document sur les calculs de complexité seront réalisés selon le paramètre  $n$  représentant le nombre de boîtes, et  $d$  est le nombre de dimensions du problème.

La seconde partie sera consacrée à la recherche de structures efficaces pour la réalisation du logiciel de visualisation. Dans cette partie, nous tenterons de répondre aux problèmes de gestion d'un grand nombre de boîtes, ainsi que ceux liés aux changement des variables de projection du pavage.

### 3.2 Boîtes et pavage

#### 3.2.1 Boîte

C'est l'entité atomique du pavage. Les accès à ses attributs sont donc cruciaux. On rappelle qu'une boîte est définie de la manière suivante :

- Un identifiant : soit des chaînes de caractères (**IDStr**) respectant un format précis (*cf.* 1.1 du document de spécifications), soit des entiers positifs (**IDInt**).
- Une liste de coordonnées dans l'ordre des variables définies en entête. Une liste de type **Interval**.
- Une liste des caractéristiques, dans l'ordre et selon les types définis en entête. Dans le pavage donné par *Realpaver* (*cf.* paragraphe [2.6.1](#)), une caractéristique est par exemple :
  - La précision (valeur numérique).
  - La caractéristique définie par les mots clefs **OUTER** et **INNER** (**String**) (*cf.* [[L.GRANVILLIERS04](#)]).
  - Le temps de calcul (valeur numérique).

Ces données seront régulièrement requises lors de la mise en œuvre des algorithmes nécessaires à la visualisation. Il est donc important que leurs accès soient rapides, voire

directs. Pour le cas de l'identifiant, s'agissant d'une simple **String** le problème de la structure à utiliser ne se pose pas. Pour la liste des coordonnées en revanche, il s'agit d'une séquence finie de données. Plusieurs possibilités sont alors envisageables :

- Un tableau : L'accès à une coordonnée est direct. Les opérations d'ajout et de suppression sont en revanche coûteuses pour les tableaux dynamiques ( $O(n)$ ).
- Une liste : Si l'accès à une coordonnée n'est pas direct ( $O(n)$ ), les opérations d'ajout et de suppression sont en temps constant.
- Une table de hachage : Coûteuse si la fonction de hachage n'est pas appropriée, une table de hachage propose cependant un accès en  $O(1)$ . Cependant le phénomène de collisions (mauvaise répartition des clefs entraînant un conflit entre deux valeurs) est à prendre en considération :

**Implémentation avec un tableau dynamique** Une telle structure permet de garantir un accès en temps constant. Cependant chaque collision va doubler l'occupation mémoire du tableau. Or même si la fonction de hachage est bonne, il est possible d'avoir au moins une collision, ce qui aurait pour conséquence une perte de l'espace mémoire qui se répercuterait sur chaque boîte.

**Gestion des collisions avec chaînage** Contrairement à la structure précédente, cette méthode permet de ne pas occuper trop d'espace en chaînant les éléments entrants en collision. Malheureusement la complexité en pire cas des opérations d'accès passe en  $O(n)$ . En revanche, grâce à une bonne fonction de hachage, on accédera généralement en temps constant sans contre coût mémoire.

Le passage en revue de ces différentes structures écarte la liste et la table de hachage implémentée par un tableau dynamique. En effet les performances des opérations d'accès de la liste ne sont pas raisonnables. De même l'implémentation d'une table de hachage par un tableau dynamique risque d'entraîner une perte de mémoire trop importante.

## Généricité des caractéristiques

Un problème majeur apparaît pour l'instanciation des boîtes. On rappelle qu'une caractéristique à plusieurs types (**String**, **Number** ou **Interval**). Plusieurs solutions sont envisageables :

- Une Map unique contenant des **String** stocke les différentes caractéristiques de la boîte. On aura casté les attributs **Number** ou **Interval** en **String**. Il sera alors nécessaire, pour chaque futur accès, d'effectuer un cast dynamique. On rajoute alors une constante supplémentaire à la complexité de cette opération.
- Trois tableaux (un pour chaque type) au sein d'une boîte. Trois Maps (une pour chaque type) « générales » au niveau du pavage. La clef d'une Map est l'id de la caractéristique et la valeur de la Map son indice dans le tableau concret. La boîte peut alors retrouver la valeur de la caractéristique au sein de son tableau.
- Chaque boîte possède trois Maps pour ces trois types de caractéristiques.



### 3.2.2 Pavage

**Problématique** Le cahier de spécification exige de l'outil la capacité à charger un pavage de taille non déterminée. Si le nombre de boîtes sera en pratique nécessairement borné (limite mémoire de la machine), il faut cependant répondre à cette attente en proposant une structure de stockage capable de supporter un très grand nombre de boîtes. De plus l'outil doit être en mesure de fournir régulièrement des listing spécifiques de boîtes. Par exemple pour afficher la liste de celles concernées par un *filtre* (cf. A.2 section 1.2, 5<sup>ème</sup> point). La structure du pavage doit être en mesure de répondre de manière efficace à des requêtes de séquences de boîtes selon un ordre particulier. La structure du pavage doit être aussi en mesure de répondre efficacement à la structure de visualisation graphique (fournir rapidement les nouvelles boîtes dans le champs de visualisation, lors d'une rotation de caméra par exemple). Quelles structures de données et quelles stratégies choisir face à de telles exigences ?

#### Étude de structures

**Vector** Collection de données à accès direct par indice. Le nombre de boîtes étant donné dans l'entête du fichier d'entrée, une implémentation par un tableau statique proposerait une complexité en temps égale à  $O(n)$  pour l'opération de stockage du pavage. Si l'accès à une boîte à partir de son indice est direct, l'opération de recherche en revanche aurait une complexité en temps égale à  $O(n)$ .

**Dictionnaire** Collection de données à accès direct par clef. Dans l'hypothèse de posséder une fonction de hachage ne provoquant pas de collisions, une implémentation par une fonction de hachage propose une complexité en temps égale à  $O(n)$  (à nouveau grâce à la connaissance du nombre de boîtes dans l'entête du fichier d'entrée).

**Arbre binaire** L'utilisation d'un arbre binaire de recherche pour la création de  $n$  boîtes aurait une complexité en temps égale à  $O(n^2)$  en pire cas. En effet il s'agit du cas où les boîtes arriveraient triées selon l'ordre inverse de celui que l'on souhaite. Ce critère d'ordonnancement des boîtes peut varier (Id, caractéristique « temps de calcul », borne inférieur d'une variable, ...). Il faudrait alors effectuer  $(p - 1)$  comparaisons, pour chaque boîte :  $\sum_{p=2}^n (p - 1)$  Soit une complexité en temps égale à  $O(n) = \frac{1}{2}n^2 - \frac{1}{2}n$ . Cependant dans le meilleur des cas, cette opération a une complexité en temps égale à  $O(n \log n)$ . La création d'un pavage composé de  $n$  boîtes à  $d$  dimensions aurait alors une complexité en temps égale à :  $O(d \times n \log(n))$ . L'arbre binaire était équilibré par définition, la complexité en temps de son opération de recherche est en  $O(\log_2 n)$  (hauteur de l'arbre).

**Arbres a-b** Il s'agit d'un arbre de recherche avec les propriétés suivantes :

- $a \leq 2$  et  $b \leq 2a - 1$  deux entiers.
- La racine a au moins 2 fils (sauf si l'arbre ne possède qu'un nœud) et au plus  $b$  fils. Les feuilles sont de même profondeur.
- Les autres nœuds internes ont au moins  $a$  et au plus  $b$  fils.

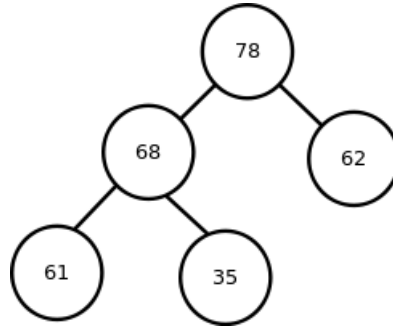


FIGURE 3.1 – arbre binaire

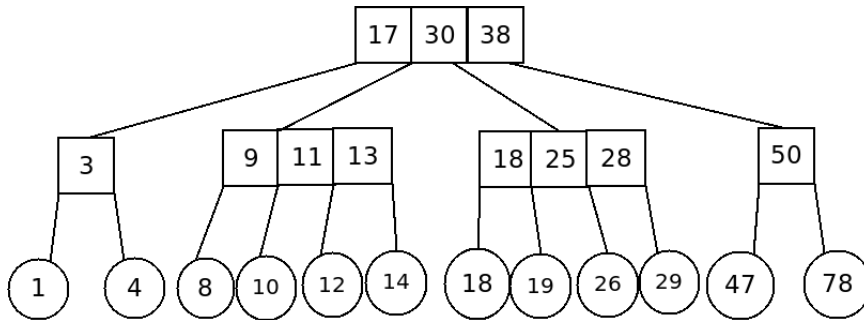


FIGURE 3.2 – a-b arbre

L'avantage des arbres a-b est que leurs hauteurs sont comprises entre les valeurs suivantes :  $\frac{\log n}{\log b} \leq h < 1 + \frac{\log n/2}{\log a}$ . Ainsi les opérations d'insertion ne seraient plus en  $O(n \log n)$  mais en  $O(\log n)$ . La création d'un pavage composé de  $n$  boîtes à  $d$  dimensions aurait alors une complexité temporelle en  $O((n \times d) \log(n))$ . La recherche d'une boîte quant à elle aurait une complexité en  $O(\log n)$ .

**Brève conclusion** Ce passage en revue de ces différentes possibilités permet, pour certaines d'entre elles, de suggérer un usage dans certaines conditions. En effet les structures telles que les *dictionnaires* ou les *vectors* ne sont pas recommandables pour le stockage du pavage. On peut notamment leur reprocher le coût trop élevé de l'opération de recherche de la structure *vector*. Quant aux *dictionnaires*, ses performances sont restrictives à l'existence d'une fonction de hachage ne provoquant pas de collisions.

Les structures arborescences ont l'avantage de pouvoir stocker et manipuler un grand nombre d'entités par des opérations ayant, la plupart du temps, des complexités en temps d'ordre logarithmique. On peut cependant envisager une utilisation de structures à accès direct (*vectors*, *dictionnaires*) en parallèle de structures arborescences dans le cadre de certaines opérations (recherche d'une liste de boîtes pour l'application d'un *filtre*, d'une vue, ...). En revanche, le maintien de la cohérence entre une structure contenant le pavage et ces structures « secondaires » introduit une redondance et donc un coût en mémoire.

## 3.3 Stockage des boîtes et visualisation

### 3.3.1 Introduction

Afin de répondre au mieux au cahier des charges du logiciel de visualisation (*cf.* Annexe A.1), trois difficultés majeures apparaissent. La première est bien entendu la gestion d'une très grande quantité de boîtes lors de l'affichage. Il est en effet nécessaire d'offrir un accès rapide aux informations des boîtes dans la fenêtre. La seconde est la gestion des filtres sur ces mêmes boîtes. Et le troisième apparaît lors du changement des variables étudiées (changement des dimensions visualisées). Dans cette section, nous chercherons d'avantage à apporter une solution pour les problèmes de temps d'accès et de changement de variables. Il est en effet probable que la gestion des filtres soit effectuée par une structure différente.

### 3.3.2 Étude d'une solution possible : le QuadTree

Une des solutions qui pourrait permettre une visualisation fluide du pavage tout en répondant au document de spécifications serait de représenter le pavages sous une forme de QuadTree pour deux dimensions ou OcTree pour trois dimensions.

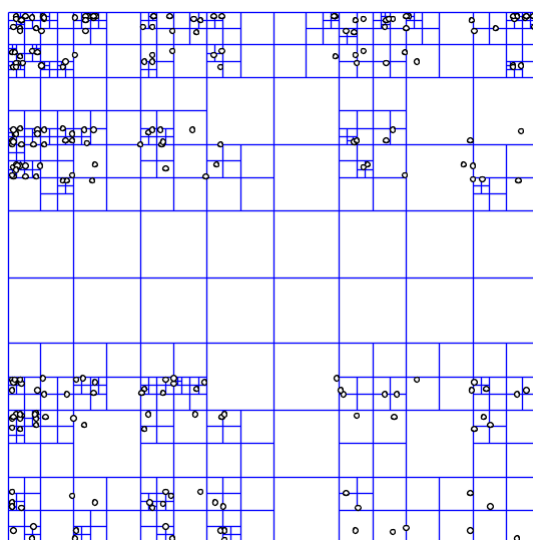


FIGURE 3.3 – Représentation d'un QuadTree où les données sont des points

Le QuadTree consiste à découper récursivement un espace fini en deux dimensions en quatre parties égales. Chacune de ces parties sont stockées dans une cellule. On itère ce mécanisme sur chacune de ces cellules jusqu'à isoler spatialement les éléments recherchés. Cette structure pourrait être utilisée pour déterminer la position des boîtes dans l'espace.

L'algorithme permettant de partitionner est récursif. Dans le cas récursif, l'algorithme divise une cellule en quatre et itère sur chacune des sous cellules. Dans le cas d'arrêt on

ne subdivise plus. Nous décrivons par la suite l'ensemble des cas de récursions et d'arrêts pour la division d'une cellule donnée. Les images jointes au texte sont des représentations des différents cas dans lesquels les cadres rouges sont des boîtes solutions et les cadres noirs une cellule du QuadTree. Vous trouverez une classification visuelle dans la table 3.1.

1. Cas d'arrêts :
  - (a) La cellule fournie est entièrement inclus dans une boîte.
  - (b) La cellule fournie contient entièrement une seule boîte.
  - (c) La cellule fournie contient en partie une seule boîte.
  - (d) La cellule fournie n'intersecte aucune boîte.
  - (e) La cellule fournie ne peut plus être subdivisée car on a fourni une taille minimale pour les espaces.
2. cas de récursions :
  - (a) La cellule fournie intersecte plusieurs boîtes.







Cas de récursion	2a. 	2a. 		
Cas d'arrêt	1a. 	1b. 	1c. 	1d. 

TABLE 3.1 – Classification visuelle des cas d'arrêts et de récursions

L'OcTree repose sur le même principe mais étendu à trois dimensions. Une cellule est donc découpée en huit parties à chaque fois.

**Avantage de cette méthode** Cette structure est particulièrement intéressante pour la visualisation du pavage. En effet pour une fenêtre de visualisation donnée, il est très simple et rapide d'extraire la sous-arborescence correspondante à la fenêtre visualisée et permet aussi de ne pas afficher les objets trop petits.

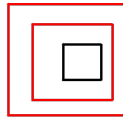


FIGURE 3.4 – Superposition de boîtes

De plus il serait possible de créer une structure reposant sur le même principe que le QuadTree mais étant  $d$ -dimensionnelle<sup>1</sup>. Chaque cellule peut alors être potentiellement subdivisée en  $2^d$  sous-cellules. Cette structure permettrait d'éviter de recalculer l'arbre à chaque changement de variables de visualisation. On évite par ailleurs le cas de superposition de boîtes pour lequel l'algorithme n'est plus efficace (cf : figure 3.4).

1.  $d$  étant la dimension du problème fourni à *Realpaver*.

**Inconvénient de cette méthode** Le problème majeur de cette méthode se présente lorsque des boîtes sont côte à côte (cf. figure 3.5). Dans une telle situation chaque

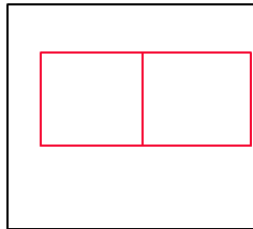


FIGURE 3.5 – État initial du QuadTree lorsque deux boîtes sont côte à côte

division va entraîner la création d'une cellule dans la même configuration. L'algorithme ne s'arrêtera donc pas avant d'avoir atteint la taille minimale pour une cellule. Nous nous retrouvons donc avec un grand nombre de cellules au niveau de ces « frontières » (cf. figure 3.6). Ainsi sachant que la précision maximale par défaut de *Realpaver* est de

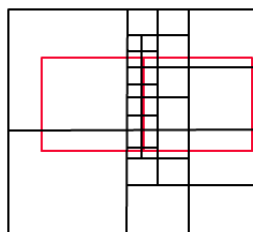


FIGURE 3.6 – État final du QuadTree lorsque deux boîtes sont côte à côte

$10^{-16}$ , cela implique qu'il est nécessaire d'au moins évaluer cette précision pour l'arbre de visualisation. Ainsi pour une sortie de *Realpaver* contenant un total  $l$  de longueurs de « frontières » cumulées et  $p$  la précision du modèle, on a un nombre de cellules à créer supérieur à  $\frac{l}{p}$  avec  $p$  très petit. Par exemple pour une sortie de *Realpaver* comportant une « frontière » de taille 1, il faudra au moins  $10^{16}$  cellules-feuilles pour la contenir.

**Brève conclusion** Dans le meilleur des cas, le QuadTree nécessite autant de cellules-feuilles que de boîtes dans le pavage. Soit  $\sum_{i=0}^{\lceil \sqrt[4]{N} \rceil} 4^i$  cellules au total. Si une cellule contient au moins un pointeur vers chacune de ces filles, un pointeur vers la cellule-mère et un pointeur vers la boîte contenue, et que l'on considère 4 octets pour stocker une adresse, on a donc une cellule de 24 octets. Donc pour un ordinateur actuel possédant une mémoire de 16 Gio, on peut stocker au maximum  $4^{14}$  ( $\approx 2 \times 10^8$ ) boîtes. Cependant on a vu qu'il était possible de se retrouver rapidement avec un QuadTree ne pouvant être stocké. Le cas cité un peu plus haut créait d'ailleurs plus de  $10^{16}$  cellules-filles pour seulement 2 boîtes.

### 3.3.3 Étude d'une seconde solution : le R-tree

Bien que le QuadTree soit une structure intéressante pour permettre l'indexation et la recherche de points dans un espace, celui-ci l'est beaucoup moins pour la gestion de données à dimensions non nulles. Le R-tree est une structure proposée en 1984 par Antonin GUTTMAN permettant l'indexation et la recherche d'éléments de dimension  $d > 0$  dans  $k$  dimensions [A.GUTTMAN84].

Le R-tree est une variante de l'arbre B, sa structure est la suivante :

- Un nœud de l'arbre correspond à une boîte non-solution du pavage. Chacun de ces nœuds est un MBR<sup>2</sup> de ces fils.
- Chaque boîte peut contenir entre  $m$  et  $M$  sous-boîtes entièrement incluses. Avec  $m \leq \frac{M}{2}$ .
- Une feuille de l'arbre est une boîte ne contenant que des boîtes solution du pavage.

La figure 3.7 donne une bonne idée de l'organisation des R-trees :

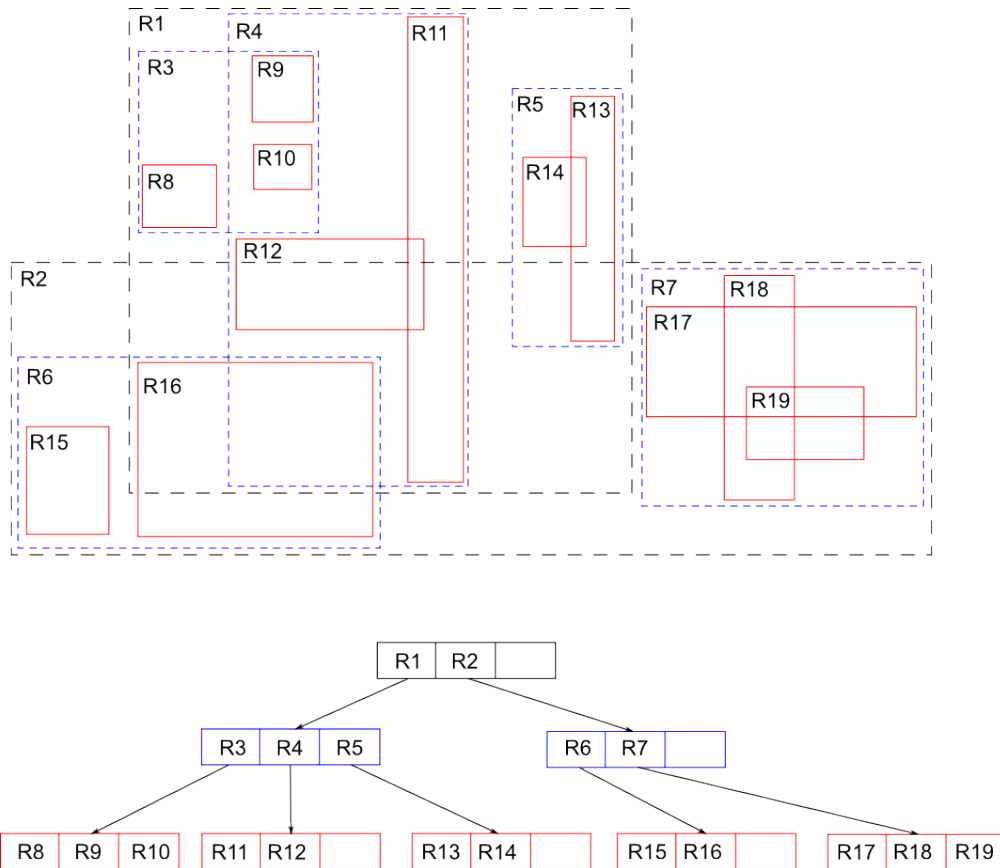


FIGURE 3.7 – Représentation d'un R-tree [Wika]

2. « Minimum bounding rectangle » : les rectangles étant, dans le cas présent,  $d$ -dimensionnels.

Les algorithmes permettant la recherche et la création du R-tree sont quant à eux décrits dans l'article de A. GUTTMAN dans la section **3. Searching and Updating**[A.GUTTMAN84]. Nous rappelons brièvement les algorithmes de recherche (cf. Algorithme 1) et d'ajout d'une boîte (cf. Algorithme 2) [YAA06]. Il est important de noter que GUTTMAN propose trois algorithmes de « split » qui permettent ensuite à l'algorithme de recherche d'être plus ou moins efficace. Du moins efficace au plus efficace pour la recherche : un algorithme linéaire, un quadratique et un exponentielle.

L'algorithme 2 d'ajout présenté est implémenté avec le split linéaire.

---

**Algorithm 1 Recherche**(nœud  $N$ , rectangle  $R$ )

---

*/\* Recherche l'ensemble des feuilles contenues dans un rectangle requête  $d$ -dimensionnel à partir d'un nœud  $N$ , les feuilles solutions seront stockées dans  $\Lambda$  \*/*

```

1: if  $N$  n'est pas une feuille then
2:   for all éléments  $n$  de  $N$  dont le MBR intersecte  $R$  do
3:     Recherche( $n$ ,  $R$ )
4:   end for
5: else { $N$  est une feuille}
6:   for all éléments  $n$  de  $N$  dont le MBR intersecte  $R$  do
7:     ajouter  $n$  à  $\Lambda$ 
8:   end for
9: end if
10: return  $\Lambda$ 

```

---

**Avantages et inconvénients du R-tree** Le R-tree est une structure de données spécialement conçue pour la recherche de données à dimensions  $d$  pour dans des espaces  $k$ -dimensionnels. L'arbre a une profondeur maximale  $h_{max}$  égale à :

$$h_{max} = \lceil \log_m n \rceil - 1 \quad (3.1)$$

et le nombre de nœuds est au pire égale à :

$$\sum_{i=1}^{h_{max}} \left\lceil \frac{n}{m^i} \right\rceil = \left\lceil \frac{n}{m} \right\rceil + \left\lceil \frac{n}{m^2} \right\rceil + \dots + 1 \quad (3.2)$$

**Brève conclusion** Afin d'avoir une idée de les performance en espace mémoire prenons  $m = 2$  et  $M = 50$  et une mémoire de 16 Gio. Une nœud du R-tree doit contenir au minimum un pointeur sur tous ces fils, un pointeur sur son père et un pointeur vers son MBR, un nœud fera donc une taille de 208 octets. On peut donc avoir jusqu'à  $8.2 \times 10^7$  nœuds, soit  $4.1 \times 10^7$  boîtes contenues dans les feuilles au total.

Bien que ne pouvant garantir de bonnes performances en pire cas<sup>3</sup>, le R-tree offre en pratique de bons résultats ; on pourra d'ailleurs se reporter à l'analyse de performances

---

3. « More than one subtree under a node may need to be searched, hence it is not possible to guarantee good worst-case performance. »[A.GUTTMAN84], section **3.1 Searching**

---

**Algorithm 2 Ajout**(élément  $E$ , nœud  $NR$ )

---

*/\* Ajoute un nouvel élément  $E$  dans le  $R$ -tree de nœud racine  $NR$  \*/*

---

- 1: On parcourt l'arbre depuis la racine  $NR$  jusqu'à la feuille appropriée. À chaque niveau, on sélectionne le nœud dont le MBR nécessite le plus petit élargissement pour contenir le MBR de  $E$
  - 2: En cas d'égalité, on choisit le nœud ayant le MBR de plus petite « superficie »
  - 3: **if** la feuille atteinte  $L$  n'est pas pleine **then**
  - 4:   On insère  $E$  dans  $L$
  - 5:   On met à jour tous les MBR de  $L$  jusqu'à  $NR$  pour qu'ils puissent contenir le MBR de  $E$
  - 6: **else** {la feuille  $L$  est pleine}
  - 7:   On définit  $\xi$  l'ensemble constitué de tous les éléments de  $L$  et du nouvel élément  $E$   
On choisit deux éléments  $e_1$  et  $e_2$  tel que la distance entre  $e_1$  et  $e_2$  soit la plus grande de toutes les autres paires de  $\xi$   
On crée deux nœuds  $l_1$  et  $l_2$  contenant respectivement  $e_1$  et  $e_2$
  - 8:   On parcourt tous les éléments restants de  $\xi$  et l'on assigne chacun d'eux à  $l_1$  ou  $l_2$ , en fonction du MBR de ces nœuds qui permet le plus petit élargissement pour contenir le MBR de l'élément considéré
  - 9:   **if** égalité de l'élargissement **then**
  - 10:     On assigne l'élément au nœud ayant le MBR de plus petite superficie
  - 11:     **if** égalité de superficie **then**
  - 12:       On assigne l'élément au nœud ayant le moins d'élément
  - 13:     **end if**
  - 14:   **end if**
  - 15:   **if** il reste  $\lambda$  éléments dans  $\xi$  et qu'un nœud contient  $m - \lambda$  éléments **then** {On rappelle que  $m$  est le nombre minimal d'éléments que peut contenir un nœud}
  - 16:     On assigne tous les éléments restants dans  $\xi$  à ce nœud
  - 17:   **end if**
  - 18:   On met à jour le MBR des nœuds se trouvant sur le chemin de  $L$  à  $NR$  pour que ceux ci couvrent le MBR de  $l_1$  et  $l_2$
  - 19:   On effectue les « splits » éventuels des nœuds supérieurs
  - 20: **end if**
-



effectuer par A. GUTTMAN<sup>4</sup>. Cependant il existe aujourd’hui de nombreuses variantes du R-tree ( R\*tree, Hilbert R-tree, etc. . . ), on pourra donc, pour une analyse plus générale des différentes implémentations, préférer *R-trees : Theory and Applications* [YAA06].

### 3.3.4 Conclusion

Au vu de l’étude comparative entre les deux structures, le R-tree nous semble une structure plus adéquate pour répondre à une visualisation des pavages. En effet, bien que le QuadTree soit adapté pour stocker des éléments sans dimension. Il devient moins performant et plus coûteux lorsque les éléments sont  $d$ -dimensionnels et notamment lorsque le pavage résultat possède un grand nombre de boîtes adjacentes comme c’est le cas pour la figure 2.4. La structure R-tree quant à elle offre de meilleurs résultats sur les objets  $d$ -dimensionnels et semble donc plus adaptée pour notre logiciel de visualisation.

---

4. section 4. Performance[A.GUTTMAN84]

## 4 Étude pratique

### 4.1 Chargement de pavages

Pour la génération des boîtes le nombre de caractéristiques est fixé à 20 et la dimension du pavage à 100. Ces valeurs ont été choisies car elles peuvent être considérées comme les valeurs au pire renvoyées par *Realpaver*. Cependant les caractéristiques pouvant être de différents types et ne connaissant pas la probabilité d'apparition de chacun de ces types, nous avons préféré construire des caractéristiques, générées aléatoirement, de chaque type en proportion un tiers (sept Number, sept String, six Interval).

Tous les résultats sont répertoriés dans la partie 4.4.1.

#### 4.1.1 Map internes

##### Pavage avec une première proposition de structure

Pour les Maps l'organisation des structures de données est la suivante :

- Il n'y a aucune structure de données globale, tout est stocké au niveau de la boîte. On a donc dans chaque boîte l'identifiant et la valeur de chaque caractéristique ou intervalle.
- Chaque boîte contient quatre Maps contenant les différentes informations :
  - Une Map pour l'ensemble des intervalles coordonnées.
  - Une Map pour les caractéristiques de type String.
  - Une Map pour les caractéristiques de type Interval.
  - Une Map pour les caractéristiques de type Number.

Les résultats obtenus sont indiqués dans la table 4.1 et 4.2 D'après les résultats, les deux structures fournissent un temps de construction et une occupation mémoire similaires. Il est encore difficile de déterminer laquelle de la TreeMap ou de la HashMap est la plus pertinente, les différences apparaîtront sans doute plus nettement au moment des tests d'accès aux caractéristiques.

##### Pavage avec une seconde proposition de structure

Pour les map, une autre organisation des structures de données possible est la suivante :

- Il y a une HashMap globale contenant le type de chaque caractéristique.
- Chaque boîte contient deux Maps contenant les différentes informations :
  - Une map pour l'ensemble des intervalles coordonnées.
  - Une liste pour l'ensemble des caractéristiques stockées dans la boîte sous la forme d'Objects.

Les résultats obtenus sont indiqués dans la table 4.3 et 4.4 Nous n'observons pas de différence nette, par rapport à la première proposition, quant au temps de construction et de l'occupation mémoire. Cette structure nécessitant d'effectuer un cast à chaque accès, il est plus avantageux de conserver la première structure.

#### 4.1.2 Listes et Tableaux internes

Pour les structures liste et tableau l'organisation des structures de données est la suivante :

- Il y a une HashMap globale contenant le type de chaque caractéristique.
- Chaque boîte contient deux listes (ou tableaux) :
  - Une liste pour l'ensemble des intervalles coordonnées.
  - Une liste pour l'ensemble des caractéristiques de la boîte stockées sous la forme d'Objects

Les résultats obtenus sont indiqués dans la table 4.5 et 4.6 On peut constater que les structures de données en liste sont bien meilleures vis-à-vis du temps de construction et de l'espace mémoire(Tout du moins pour l'ArrayList) que les structures map de la première partie. Étrangement la structure LinkedList demande un espace mémoire presque 60% plus grand que celui alloué pour l'ArrayList.

#### 4.1.3 Maps globales

Les boîtes sont composées de 4 ArrayLists :

- Une ArrayList pour l'ensemble des intervalles coordonnées.
- Une ArrayList pour les caractéristiques de type String.
- Une ArrayList pour les caractéristiques de type Interval.
- Une ArrayList pour les caractéristiques de type Number.

Il faut cependant savoir à quels indices se trouvent les différents éléments pour effectuer des accès directs. Pour cela, on dispose ici de quatre Maps globales :

- Une Map pour l'ensemble des intervalles coordonnées.
- Une Map pour les caractéristiques de type String
- Une Map pour les caractéristiques de type Interval
- Une Map pour les caractéristiques de type Number

La composition de chacune de ces maps est la suivante :

**Clef :** ID de la coordonnée ou de la variable.

**Valeur :** Indice de l'objet dans le tableau de la boîte.

**Observations :** On propose trois cas de tests où les Maps globales seront :

- Des HashMaps : 4.7
- Des TreeMaps : 4.8

Les trois cas de tests nous renvoient des résultats relativement similaires mais qui se montrent aussi performants que les résultats de l'ArrayList. Il s'agit donc d'une organisation intéressante à étudier au niveau des temps d'accès.

## 4.2 Accès au élément d'une boîte

Dans cette partie nous allons effectuer des tests d'accès aux éléments d'une boîte. Le nombre de coordonnées et de caractéristiques sont les mêmes que dans la partie 4.1 c'est-à-dire 100 valeurs pour les coordonnées et 20 caractéristiques. Parmi ces coordonnées, sept sont de type Number, sept de type String et 6 de type Interval. Pour toutes les structures testées, il est nécessaire d'avoir une structure globale contenant les types de chaque caractéristique. Nous effectuerons d'abord des tests d'accès aux caractéristiques puis des tests d'accès aux coordonnées. Nous rappelons aussi que les éléments sont générés aléatoirement durant la création.

Pour plus d'aisance, nous considérerons que les identifiants sont de types Integer. Un type String revenant au final au même mais nécessitant au préalable un hashage.

Il est aussi important de préciser que, pour tester, nous avons créé dix-mille boîtes et que pour accéder à une caractéristique, nous accédons d'abord à une boîte aléatoirement. Les boîtes sont stockées dans un tableau de taille statique à accès direct par indice. Même s'il est vrai que cette méthode introduit une constante dans le calcul de la complexité, elle permet de ne pas toujours accéder à la même information et donc de ne pas permettre un accès plus rapide qui ne serait pas réaliste.

Tous les résultats sont répertoriés dans la partie 4.4.2.

### 4.2.1 Tests d'accès aux caractéristiques

Pour les tests d'accès aux caractéristiques, nous générons aléatoirement un nombre donnant l'identifiant de la caractéristique.

Les résultats d'accès pour des boîtes contenant une Map (HashMap ou TreeMap) par type de caractéristique (Number, String ou Interval) sont indiqués dans le tableau 4.9.

On peut voir que les deux structures ne se distinguent pas beaucoup en terme de performance, on observe seulement un écart de moins d'une seconde entre la HashMap et la TreeMap à partir de cent millions d'accès.

Les résultats d'accès pour des boîtes contenant une Map (HashMap ou TreeMap) pour stocker toutes les caractéristiques sont indiqués dans le tableau 4.10.

Cette fois-ci les résultats sont légèrement différents pour les deux structures. La HashMap est en effet plus efficace que le TreeMap de presque 10%, elle est d'ailleurs plus efficace que lorsque chaque caractéristique était stockée dans sa propre Map.

Les résultats d'accès pour des boîtes contenant une ArrayList ou une LinkedList pour stocker toutes les caractéristiques sont indiqués dans le tableau 4.11.

On peut nettement voir l'efficacité de la structure ArrayList pour les temps d'accès par rapport à la LinkedList, mais aussi par rapport à toutes les autres structures précédemment proposées. Celle-ci étant plus efficace de 40% à 50% par rapport aux autres structures. On peut aussi s'étonner des performances de la LinkedList par rapport aux structures Maps, mais on peut supposer que leur proximité est due au faible nombre de caractéristiques à stocker.

Les résultats d'accès pour des boîtes contenant une ArrayList pour chaque caractéristique, avec une Map globale pour retrouver les indices de chacune de ces caractéristiques dans les boîtes, sont indiqués dans le tableau 4.12.

D'après les résultats, l'utilisation d'une HashMap globale est environ 10% plus efficace que d'utiliser une TreeMap globale. La structure proposée avec une HashMap globale offre de bonnes performances par rapport aux autres structures proposées précédemment, mais reste tout de même moins efficace qu'une ArrayList stockant toutes les caractéristiques.

**Conclusion sur l'accès aux caractéristiques** Les meilleures performances sont obtenues lorsqu'une boîte ne contient qu'une ArrayList pour stocker toutes les caractéristiques. Cependant on peut se demander s'il peut s'avérer utile d'obtenir uniquement les caractéristiques pour un type donné. Si c'est le cas, il sera nécessaire d'effectuer une recherche au préalable pour obtenir les éléments désirés. Tandis que lorsque les boîtes contiennent une ArrayList pour chaque caractéristique, bien que les performances d'accès unique soient légèrement moins bonnes, cette recherche serait alors inutile.

#### 4.2.2 Tests d'accès aux coordonnées

Les résultats d'accès aux coordonnées pour différentes structures (Map et List) sont détaillés dans le tableau 4.13.

Sans surprise l'ArrayList reste la structure la plus adaptée pour stocker les coordonnées.

### 4.3 Conclusion

Au vu des résultats sur les différentes architectures, l'architecture proposant que chaque boîte ne contiennent qu'une ArrayList pour stocker toutes les caractéristiques est la plus efficace, que ce soit au niveau des temps d'accès qu'au niveau des temps de construction et d'espace mémoire. Cependant l'architecture où les boîtes contiennent une ArrayList pour chaque caractéristique offre des résultats moins bons mais relativement proches, et permet une meilleure performance pour l'accès aux éléments d'un même type. Il sera donc nécessaire de voir si une telle action est utile.

Pour le stockage des coordonnées, l'utilisation de l'ArrayList est préconisé, les résultats la dégageant nettement du lot.

## 4.4 Annexes

### 4.4.1 Études de performance de création d'une boîte

#### Études de performance avec des Maps

Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
100	0.05s	52Mo	1Mo
1000	0.09s	52Mo	7Mo
10000	0.16s	119Mo	70Mo
100000	0.86s	776Mo	667Mo
1000000	Out of memory		

TABLE 4.1 – Étude de performances avec des HashMap

Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
100	0.06s	52Mo	1Mo
1000	0.07s	52Mo	6Mo
10000	0.16s	138Mo	64Mo
100000	0.84s	776Mo	635Mo
1000000	Out of memory		

TABLE 4.2 – Étude de performances avec des TreeMap

Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
100	0.06s	52Mo	1Mo
1000	0.08s	52Mo	7Mo
10000	0.13s	118Mo	69Mo
100000	0.84s	776Mo	655Mo
1000000	Out of memory		

TABLE 4.3 – Étude de performances avec des HashMap seconde structure

Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
100	0.07s	52Mo	1Mo
1000	0.08s	52Mo	6Mo
10000	0.15s	138Mo	64Mo
100000	0.88s	776Mo	633Mo
1000000	Out of memory		

TABLE 4.4 – Étude de performances avec des TreeMap seconde structure

## Études de performance avec des Tableaux et des Listes

Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
100	0.05s	52Mo	<1Mo
1000	0.06s	52Mo	4Mo
10000	0.09s	66Mo	37Mo
100000	0.30s	408Mo	342Mo
1000000	Out of memory		

TABLE 4.5 – Étude de performances avec des ArrayList

Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
100	0.05s	52Mo	1Mo
1000	0.08s	52Mo	5Mo
10000	0.1s	88Mo	54Mo
100000	0.30s	776Mo	542Mo
1000000	Out of memory		

TABLE 4.6 – Étude de performances avec des LinkedList



## Études de performance avec des maps globales

Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
100	0.05s	52Mo	0Mo
1000	0.07s	52Mo	4Mo
10000	0.1s	66Mo	38Mo
100000	0.34s	424Mo	351Mo
1000000	Out of memory		

TABLE 4.7 – Étude de performances avec des HashMap globales

Nombre de boîtes	CPU	Mémoire totale	Mémoire utilisée
100	0.05s	52Mo	0Mo
1000	0.06s	52Mo	4Mo
10000	0.11s	66Mo	38Mo
100000	0.38s	422Mo	350Mo
1000000	Out of memory		

TABLE 4.8 – Étude de performances avec des TreeMap globales

#### 4.4.2 Études de performance d'accès aux éléments d'une boîte

##### Comparaison des performances d'accès aux caractéristiques

Structure \ Nombre d'accès	$10^6$	$10^7$	$10^8$
HashMap	0.31s	2.86s	28.56s
TreeMap	0.30s	2.82s	27.84s

TABLE 4.9 – Relevé des temps CPU d'accès aux caractéristiques en secondes pour la HashMap et la TreeMap

Structure \ Nombre d'accès	$10^6$	$10^7$	$10^8$
HashMap	0.28s	2.67s	26.34s
TreeMap	0.32s	3.01s	29.84s

TABLE 4.10 – Relevé des temps CPU d'accès aux caractéristiques en secondes pour la HashMap et la TreeMap seconde structure

Structure \ Nombre d'accès	$10^6$	$10^7$	$10^8$
ArrayList	0.19s	1.58s	15.12s
LinkedList	0.31s	2.79s	27.59s

TABLE 4.11 – Relevé des temps CPU d'accès aux caractéristiques en secondes pour l'ArrayList et la LinkedList

Structure \ Nombre d'accès	$10^6$	$10^7$	$10^8$
HashMap	0.20s	1.81s	17.49s
TreeMap	0.22s	1,99s	19,48s

TABLE 4.12 – Relevé des temps CPU d'accès aux caractéristiques en secondes pour l'architecture avec des maps globales

### Comparaison des performances d'accès aux coordonnées

Structure \ Nombre d'accès	$10^6$	$10^7$	$10^8$
HashMap	0.36s	3.52s	34.98s
TreeMap	0.54s	5.35s	53.06s
ArrayList	0.27s	2.57s	25.46s
LinkedList	0.57s	5.61s	55.50s

TABLE 4.13 – Relevé des temps CPU d'accès aux caractéristiques en secondes pour l'accès aux coordonnées

## 5 Conclusion

L'étude proposée dans ce document apporte des éléments de réponses à des problèmes de conception majeurs de l'outil de visualisation. Nous proposons donc une étude des structures permettant l'accès aux éléments d'une boîte, ainsi qu'une étude préliminaire sur les structures stockant le pavage, avec pour objectif de répondre au mieux au cahier des charges. Pour l'étude d'accès aux éléments, différents tests Java ont été effectués, et mettent en avant l'efficacité de l'ArrayList pour stocker ces éléments. Nous avons aussi tenté de proposer une structure efficace pour permettre une visualisation fluide. Le R-tree semble correspondre aux attentes du logiciel, cependant il est toujours nécessaire d'effectuer une seconde étude pour trouver quelle implémentation du R-tree est la plus pertinente.

Bien que nous apportons de nombreux éléments de réponses aux problèmes de conception, il sera encore nécessaire de faire une étude approfondie sur la gestion des filtres, qui a malheureusement dû être mis de côté lors de cette étude. Il est en effet nécessaire de définir comment trouver ou organiser les boîtes en fonction des filtres à appliquer. La gestion des filtres a aussi un impact sur la structure de visualisation puisque ceux-ci vont jouer un rôle prépondérant sur l'affichage final (ordre d'affichage, couleur des boîtes...). De même, nous n'avons effectué aucune étude sur un moyen de sauvegarder le pavage.

# A Annexes

## A.1 Cahier des charges

# Cahier des charges d'un outil de visualisation de pavages

5 mai 2011

**Introduction :** L'objectif est de définir un outil de visualisation d'une projection en une, deux ou trois dimensions d'un pavage à N dimensions typiquement produit par un outil de résolution par intervalles de problèmes numériques (tel que Realpaver). Cet outil permettra de visualiser entièrement ou partiellement le pavage et d'ajuster les propriétés graphiques de la visualisation, puis d'exporter le résultat. Une indépendance vis à vis du logiciel fournissant les données est requise. Le traducteur pour passer du format de sortie du logiciel source au langage d'entrée dédié à l'outil de visualisation n'entre pas dans la conception.

## 1 Format d'entrée

### 1.1 Entête

- Nombre de boîtes. Cette valeur pourra évoluer (affichage dynamique réagissant à chaque nouvelle donnée fournie)
- Liste des noms des variables (nombre de variables implicite)
- Liste des noms et types (nombre, chaîne) des caractéristiques associées aux boîtes (e.g., temps de calcul, précision, certification, ...)
- Autres données spécifiques à l'outil (e.g., filtres, affichages conditionnels, points de références ... )

### 1.2 Corps

Ensemble de boîtes comprenant chacune :

- Identifiant unique
- Coordonnées : un intervalle de la forme  $[a,b]$  (a et b sont des nombres flottants) pour chaque variable
- Caractéristiques : une liste de valeurs (types définis en entête)

## 2 Fonctionnement

- Affichage conditionnel : définir un style d'affichage paramétrable (e.g., couleur, transparence, trait, ...) en fonction de conditions sur la boîte

(e.g., l'intervalle de la variable  $x$  chevauche l'intervalle  $[-10, 0]$  ; la boite contient le point de référence  $P$  ; la caractéristique "temps de calcul" est comprise entre 0 et 10 ; ...)

- Filtrage : n'afficher que les boites vérifiant certaines conditions (cf. exemples ci-dessus)
- Choix des une, deux ou trois dimensions à afficher
- Définition des points de références éventuellement associés à des labels
- Ajout de commentaires aux boîtes et aux points de références
- Sauvegarde
- Exportation :
  - Image bitmap (png, jpeg, ...) ou vectorielle (svg, eps, ...)
  - Textuelle : identique au format d'entrée

Dans les deux cas, on pourra choisir s'il l'on veut l'intégralité des données, celles correspondantes à la fenêtre de visualisation ou juste celles de la selection

### 3 Visualisation

- Zoom : zoomer/dé-zoomer ; revenir à une vue globale du pavage (filtré)
- Déplacement (1, 2 ou 3D) de la fenêtre de visualisation
- Pour la 3D : rotation "haut/bas" et "gauche/droite". En 1D et 2D : pas de rotation
- Possibilité de définir textuellement les zooms, déplacements et rotations.
- Sélection : une entité ( ou un ensemble ) avec possibilité d'agir dessus ( changer sa couleur, la visualiser dans une nouvelle fenêtre, modifier son label ... )
- Informations numériques :
  - Globales : état de la mémoire, nombre de pavés total/filtrés/affichés, Id de l'objet pointé, coordonnées du pointeur
  - Spécifiques : pour une boîte donnée ( ou un point de référence ), pouvoir afficher **toutes** les données associées (Id, coordonnées, caractéristiques et style d'affichage)



## A.2 Documents de spécifications

# Spécification d'un outil de visualisation de pavages

JERMANN.C MARGUERITE.A

1<sup>er</sup> mars 2012

**Introduction :** L'objectif est de définir un outil de visualisation d'une projection en une, deux ou trois dimensions d'un pavage à N dimensions typiquement produit par un outil de résolution par intervalles de problèmes numériques (tel que Realpaver). Cet outil permettra de visualiser entièrement ou partiellement le pavage et d'ajuster les propriétés graphiques de la visualisation, puis d'exporter le résultat. Une indépendance vis à vis du logiciel fournissant les données est requise. Le traducteur pour passer du format de sortie du logiciel source au langage d'entrée dédié à l'outil de visualisation n'entre pas dans la conception.

**Mise en garde** Il existe des différences entre les images et le texte. Ces différences seront signaler par le symbole  $\triangle$ . Dans tous les cas se référer en priorité au texte

## 1 Format d'entrée

### 1.1 Généralités

- Une entrée = un pavage ; l'entrée peut-être dynamique si le pavage est en cours de production.
- Format : texte au codage UTF-8
- L'entrée comporte une partie entête et une partie corps ; l'entête se place obligatoirement avant le corps.
- Les identifiants des données sont soit des chaînes de caractères (IDStr) restreintes aux lettres, aux chiffres et à l'underscore et commençant obligatoirement par une lettre (e.g., variables, caractéristiques, ...), soit des entiers positifs (IDInt) (e.g., boîtes, points de référence, ...). Dans tous les cas, l'identifiant est unique.
- Les types des données manipulées sont : Number, Interval et String.
- Les valeurs de type String sont des chaînes de caractères comprises entre guillemets (") et pouvant contenir tout caractère à l'exception des guillemets.
- Les données de type Number sont représentées par des flottants (cf. norme IEEE 754).
- Les données de type Interval sont représentées par deux données de type Number.

### 1.2 Entête

Les éléments marqués d'un astérisque peuvent être omis.

- Nombre de boîtes (entier positif, type Number). Lorsque l'entrée est dynamique, cette valeur est inconnue (remplacée par un tag ou une valeur spécifique, e.g., 0).
- Liste des identifiants (IDStr) des variables (ordre important).
- (\*) Liste des identifiants (IDStr) et types (Number, Interval ou String) des caractéristiques (e.g., temps de calcul, précision, certification) associées aux boîtes (ordre important).
- (\*) Liste des points de références, chacun composé de :
  - identifiant (IDInt)
  - label (court texte, type String)
  - commentaire (texte plus long, optionnel, type String)
  - coordonnées (autant que de variables, type Number)
  - caractéristiques d'affichage :

- . Couleur : code RVB (type String)
- . Taille des points (en pixels, type Number)
- . Police du label (type String) et son style (gras, italique, gras-italique)
- (\*) Liste des filtres. Un filtre détermine un ensemble de boîtes à afficher. Les filtres sont considérés en disjonction : toute boîte satisfaisant au moins un filtre est affichée. Chaque filtre se définit au moyen d'une règle, semblable à celle composant un affichage conditionnel. Chaque filtre est composé de :
  - identifiant (IDInt)
  - label (court texte, type String)
  - règle : liste de conditions (considérées en conjonction) ; chaque condition est composée de :
    - . un élément testé : l'identifiant (IDInt) d'une boîte, la valeur (Interval) d'une variable (IDStr), la valeur (selon son type) d'une caractéristique (IDStr), ou une formule (Interval) combinant des variables au moyen d'opérations mathématiques.
    - . une relation, selon le type d'élément testé
      - IDInt boîte :  $=, \neq, <, \leq, >, \geq$
      - Interval :  $\ni, \not\ni, <_{inf|sup}, \leq_{inf|sup}, >_{inf|sup}, \geq_{inf|sup}$
      - String :  $=, \neq, \ni, \not\ni$
    - . une valeur comparative, selon le type d'élément testé
      - IDInt boîte : un entier positif
      - Interval : une valeur de type Number
      - String : une valeur de type String
- (\*) Liste des affichages conditionnels. Un affichage conditionnel définit comment doivent apparaître les boîtes vérifiant certaines conditions. Les affichages conditionnels sont considérés en disjonction et dans l'ordre de leur définition (toute boîte s'affiche selon les modalités du premier affichage conditionnel dont elle remplit les conditions). Chaque affichage conditionnel est composé de :
  - identifiant (IDInt)
  - label (court texte, type String)
  - règle (cf. Filtres)
  - style, définissant les modalités d'affichage des boîtes validant la règle, c'est-à-dire sa couleur (avec transparence, code RVBA, type String)
- (\*) Définition du point de vue :
  - Liste des 1, 2 ou 3 variables visualisées
  - Les coordonnées de la caméra définissant le mapping  $1/2/3D \rightarrow 2D$  (cf. [http://en.wikipedia.org/wiki/3D\\_projection](http://en.wikipedia.org/wiki/3D_projection)) ; sera défini précisément une fois la technologie utilisée pour la visualisation déterminée.

### 1.3 Corps

Liste de boîtes du pavage, chacune comprenant :

- identifiant (IDInt)
- Liste des coordonnées (type Interval) de chaque variable dans la boîte, dans l'ordre dans lequel les variables sont listées dans l'entête.
- Liste des caractéristiques, dans l'ordre et selon les types définis en entête

## 2 Présentation de l'IHM générale

**Note sur les informations de spécifications** Les spécifications seront dans cette partie décrites textuellement et illustrées. Les illustrations peuvent être incomplètes et il est nécessaire de s'appuyer sur le texte pour les spécifications exactes. Dans la mesure du possible les éléments manquant dans les illustrations seront annotés par  $\triangle$ . On trouvera 5 composantes dans la fenêtre de l'outil :

- Une barre de menu en haut.

- Une barre d'outils située sous la barre de menu.
- Une partie centrale de visualisation.
- Un panneau latéral-droit d'édition et d'information.
- Une barre d'état en bas.

**Remarque :** Lors du redimensionnement de la fenêtre de l'application, seule la partie centrale de visualisation sera redimensionnée; le panneau latéral aura une largeur fixe et une hauteur minimale qui définira aussi la hauteur minimale de la fenêtre d'application. Ni la partie centrale ni le panneau latéral ne seront dotés de barres de défilement.

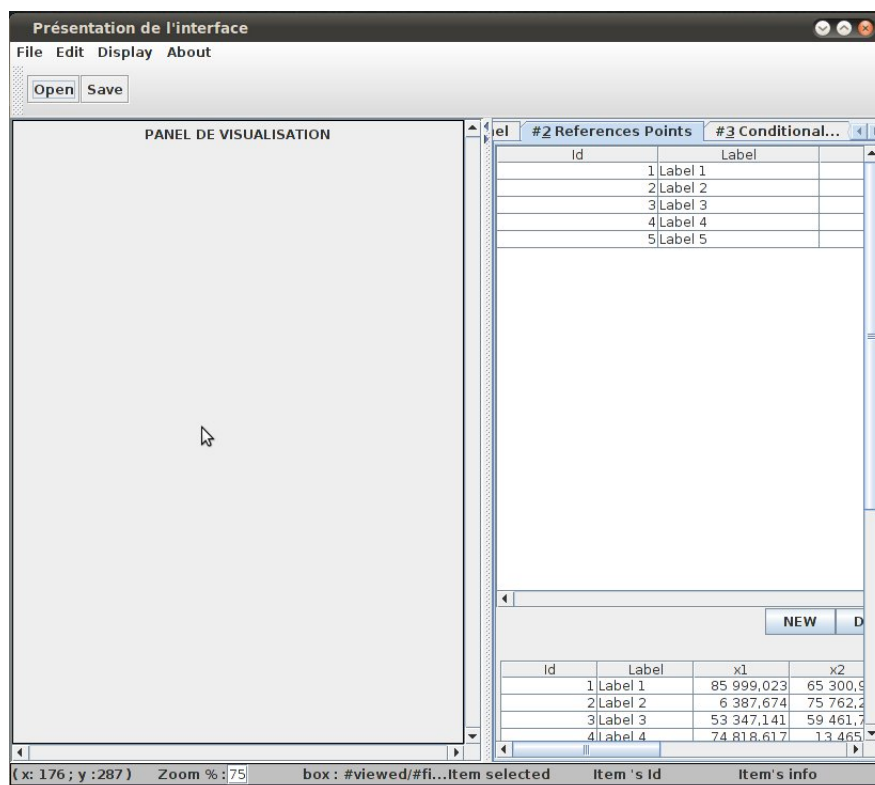


FIGURE 1 – IHM générale

## 2.1 Barre de menu

Elle est composée des sous-menus :

- File :
  - Open
  - Save
  - Export as image
  - Quit
- Edit :
  - Preferences
  - Create a Ref. Point
  - Select
    - . All
    - . by Id
    - . by Coord.
- Display
  - Choose variables
  - Zoom
    - . All
    - . In
    - . Out
- Help :△
  - Help
  - About

## 2.2 Barre d'outil

Elle est composée d'icônes (raccourcis) d'items de la barre de menu décrite précédemment. Son contenu est paramétrable par l'utilisateur. Les éléments suivants y sont toujours présents :

- les ID des variables visualisées (3 combobox éditables comprenant chacun la liste de toutes les variables). $\Delta$
- Le zoom homogène, i.e., égal sur toutes les dimensions (1 champ texte éditable et 2 boutons "+" et "-"). $\Delta$

## 2.3 Menu contextuel(à compléter)

Menu s'ouvrant lors d'un clic du bouton droit de la souris dans la zone de visualisation. Il contient les éléments suivants :

- Create a Ref. Pt
- Zoom All
- Zoom In

## 2.4 Panneau de visualisation

C'est ici que l'on observera la représentation graphique 1, 2 ou 3D du pavage. Il sera interactif avec la souris et le clavier (cf partie 5).

## 2.5 Panneau latéral

Il est composé de 4 onglets :

- Box (cf partie 3.1)
- Reference Points (cf partie 3.2)
- Filters (cf partie 3.3)
- Conditional Style (cf partie 3.4)

## 2.6 Barre d'état

Elle est contient divers information générales sur les données de l'application :

- L'ID de l'item survolé.
- Les coordonnées de la souris dans le panneau de visualisation.
- Le nombre de boites vues/filtrées/total.

# 3 Panneau latéral

## 3.1 Onglet Box

Essentiellement dédié à la lectures d'informations complémentaires de la visualisation. Il contient 4 parties :

- Dans la partie supérieure, une liste des boites donnant leurs ID, les valeurs (intervalles) des variables visualisées (1, 2 ou 3), la liste des ID des filtres dont la règle est validée par la boite  $\Delta$ , l'ID de l'affichage conditionnel qui s'applique à la boite  $\Delta$ . Cette liste pourra ne comporter que les boites actuellement visualisées, que celles satisfaisant au moins un filtre, ou l'intégralité des boites du pavage en entrée (choix au moyen d'un radio-bouton $\Delta$ ). Elle pourra être réordonnée, de façon croissante ou décroissante, selon les ID ou les valeurs des variables visualisées (selon la borne inférieure de l'intervalle). Si la liste excède la taille de la zone dédiée, une barre de défilement verticale apparaît. La sélection, par clic gauche de la souris, d'une boite de la liste la met également en surbrillance dans la fenêtre de visualisation. Réciproquement, la sélection d'une boite dans la zone de visualisation sélectionne automatiquement celle-ci dans la liste.

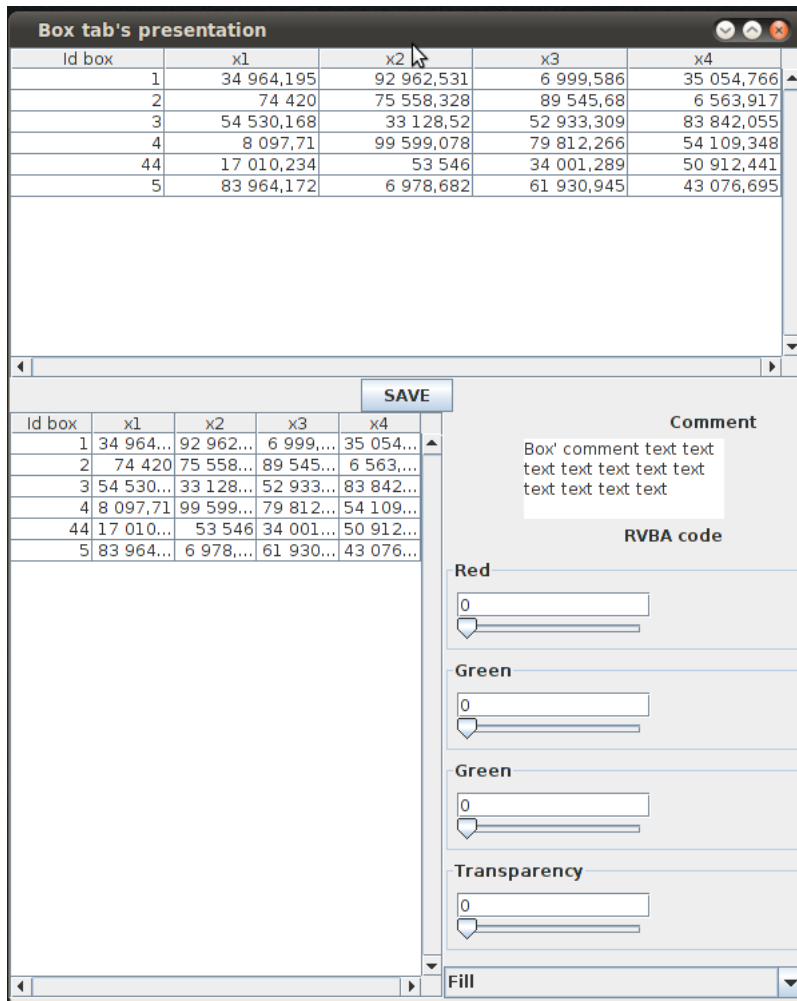


FIGURE 2 – Onglet de visualisation des caractéristiques des boîtes

- Dans la partie inférieure gauche, les informations (non éditables) détaillées de la boîte sélectionnée dans la zone supérieure : la liste des valeurs de toutes les variables, munie au besoin d'un barre de défilement verticale, ordonnable (croissant ou décroissant) selon les noms des variables ou leurs valeurs (bornes inférieures ou supérieures) ; et la liste des caractéristiques de la boîte, munie au besoin d'un barre de défilement verticale, ordonnable (croissant ou décroissant) selon les noms des caractéristiques, ou leurs valeurs (d'abord les String, puis les Number, puis les Interval selon leur borne inférieure).
  - Dans la partie inférieure droite, le commentaire et la caractéristiques d'affichages de la boîte actuellement sélectionnée dans la zone supérieure :
    - l'ID et le label de l'affichage conditionnel s'appliquant à la boîte.  $\Delta$
    - 4 glissières associées à des champs textes permettant de personnaliser la couleur et la transparence de la boîte selon le codage RVBA.
- SUPPRIMER LE COMBO ET LA NOTION DE COULEUR DE TRAIT.
- Un bouton Save permet d'enregistrer le style d'affichage personnalisé et le commentaire saisi.

### 3.2 Onglet Reference Points

**Définition :** Un point de référence constitue un point de repère visuel, éventuellement associé à un court texte (label), que l'utilisateur peut placer dans la fenêtre de visualisation. Si le pavage

a N variables, le point de référence a N coordonnées. Les points de référence peuvent être gérés (créés, modifiés, supprimés) depuis l'onglet éponyme. Il peuvent aussi être créés dans la fenêtre de visualisation via le menu contextuel. Dans ce cas, le panneau latéral basculera automatiquement sur l'onglet de points de référence dans lequel le nouveau point créé sera sélectionné ; ses coordonnées prendront des valeurs par défaut (paramétrables dans les préférences de l'application) sauf celles correspondant aux variables visualisées qui prendront pour valeur les coordonnées du pointeur de la souris au moment de la création. Dans tous les cas, les points de références reçoivent un identifiant (IDInt) unique à leur création et celui-ci n'est pas modifiable, à l'inverse de toutes les autres caractéristiques des points de référence.

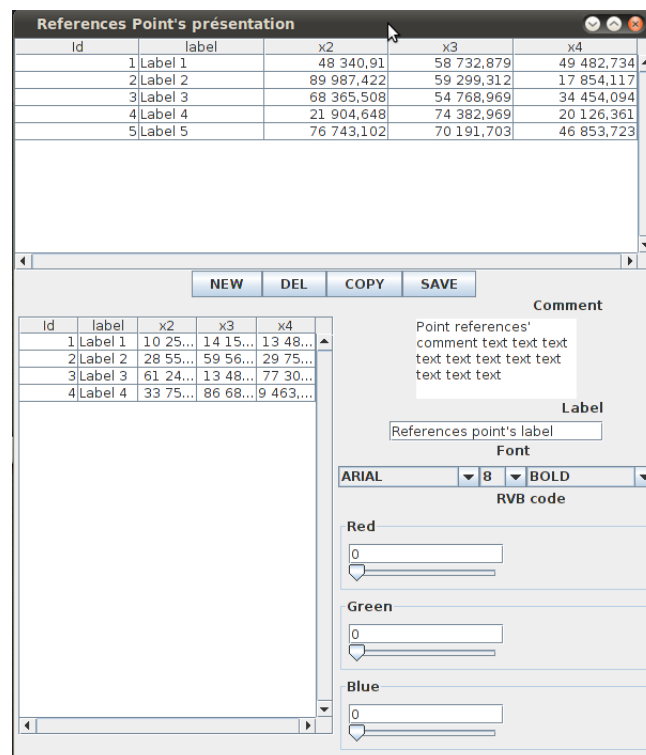


FIGURE 3 – Onglet d'édition des Points de références

### Utilisation de l'onglet :

- Dans la partie supérieure, la liste des points de référence avec leur ID, leur label et leurs coordonnées pour les variables visualisées. Cette liste pourra ne comporter que les points actuellement visualisées, ou l'intégralité des points définis (choix au moyen d'un radio-bouton  $\Delta$ ). Elle pourra être réordonnée, de façon croissante ou décroissante, selon les ID, les labels, et les valeurs des variables visualisées. Si la liste excède la taille de la zone dédiée, une barre de défilement verticale apparaîtra. La sélection, par clic gauche de la souris, d'un point de la liste le met également en surbrillance dans la fenêtre de visualisation. Réciproquement, la sélection d'un point dans la zone de visualisation sélectionne automatiquement celui-ci dans cette liste.
- Quatre boutons séparant la partie supérieure de la partie inférieure :
  - . New : Pour créer un nouveau point de référence. Il aura tous ces champs rempli de valeurs par défaut (paramétrables dans le menu Préférences).
  - . Del : Pour supprimer le point de référence sélectionné.
  - . Copy : Pour créer un nouveau point de références avec les même valeurs que celui sélectionné.

- . Save : Pour sauvegarder les modifications apportées dans la partie inférieure au point actuellement sélectionné.
- Dans la partie inférieure gauche, les coordonnées complètes (éditables) du point sélectionné dans la zone supérieure : la liste des valeurs de toutes les variables, munie au besoin d'une barre de défilement verticale, ordonnable (croissant ou décroissant) selon les noms des variables ou leurs valeurs.
- Dans la partie inférieure droite, le label (éritable), un commentaire (éritable) et les caractéristiques d'affichage du point actuellement sélectionné :
  - . La police utilisée pour le label
    - Le nom de la police
    - La taille de la police
    - La mise en forme (gras, souligné, italique)
  - . 3 glissières associées à des champs textes permettant de personnaliser la couleur selon le codage RVB.
  - . 1 glissière associée à un champ texte permettant de régler la taille du point de 1 à 10 pixels de diamètre.  $\triangle$

### 3.3 Onglet Filters

**Définition :** Un filtre (cf. partie 1.2) permet de définir une règle que doivent valider les boîtes qui seront affichées. L'ensemble des filtres sont considérés en disjonction, c'est à dire que toute boîte validant la règle d'au moins un filtre est affichée. La règle d'un filtre est une conjonction de conditions. Une condition porte sur n'importe quel élément d'une boîte : son ID, les valeurs de ses variables, ses caractéristiques ; elle peut également être formulée comme une expression mathématique sur les variables de la boîte (évaluée par arithmétique d'intervalles). La condition prend la forme d'une comparaison entre l'élément choisi et une valeur de référence. Les relations de comparaisons possibles et le type de la valeur de référence dépendent de l'élément choisi (cf. partie 1.2).

Les filtres sont gérés (créés, modifiés, supprimés) exclusivement depuis l'onglet Filters. Ils reçoivent un identifiant (IDInt) unique à leur création et celui-ci n'est pas modifiable, à l'inverse de toutes leurs autres caractéristiques.

#### Utilisation de l'onglet :

- Dans la partie supérieure, la liste des filtres avec leur ID, leur label (sous forme de champ texte directement éritable dans la liste), leur état  $\triangle$  (actif ou inactif, sous forme de checkbox directement éritable dans la liste) et le nombre de boîtes qui valident leurs règles  $\triangle$ . Cette liste pourra être réordonnée, de façon croissante ou décroissante, selon les ID, les labels, l'état et le nombre de boîtes. Si la liste excède la taille de la zone dédiée, une barre de défilement verticale apparaîtra.
- Quatre boutons séparant la partie supérieure de la partie inférieure :
  - . New : Pour créer un nouveau filtre vierge.
  - . Del : Pour supprimer le filtre sélectionné.
  - . Copy : Pour créer un nouveau filtre avec les mêmes valeurs que celui sélectionné.
- Dans la partie inférieure, deux listes côte à côte :
  - . la liste des conditions constituant la règle du filtre. Cette liste indique, pour chaque condition, l'élément testé, la relation de comparaison choisie, et la valeur de référence. La sélection d'une condition dans cette liste remplit les champs éditables se trouvant sous la liste avec les données correspondantes.
  - . la liste des ID des boîtes validant la règle du filtre. La sélection d'une boîte dans cette liste la sélectionne automatiquement dans l'onglet Box et la met en surbrillance dans la fenêtre de visualisation.

Sous la liste des conditions se trouvent des champs éditables permettant d'ajouter, modifier et supprimer une condition :  $\triangle$

- . Un combobox pour le choix de l'élément testé (ID, variable, caractéristique, formule).



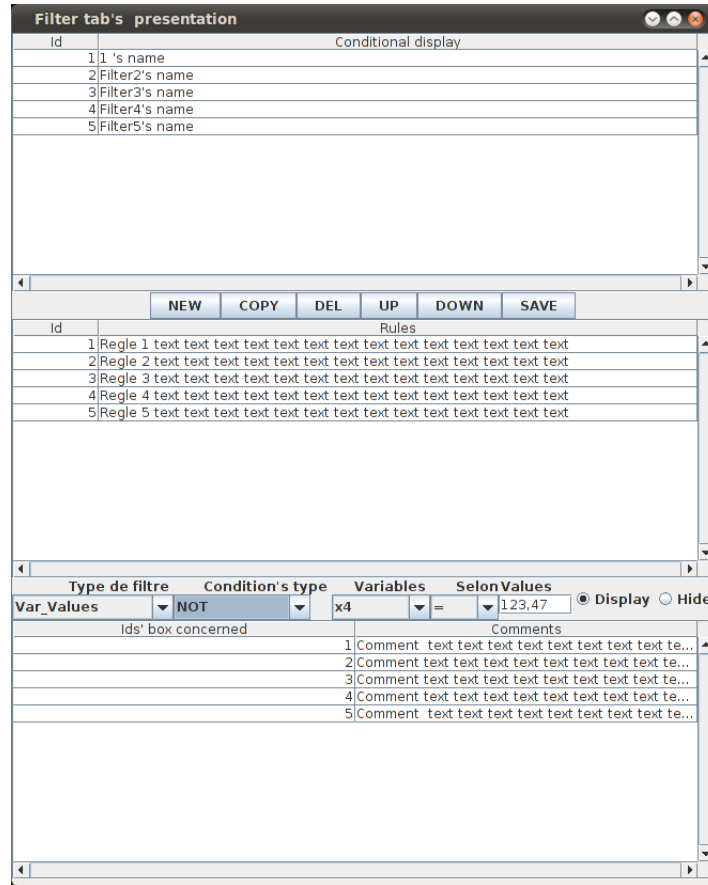


FIGURE 4 – Onglet de gestion des filtres

- . Une second combobox, permettant de choisir l'ID de la variable ou de la caractéristique concernée ; il est remplacé par un champ texte permettant de saisir la formule ; ou par rien du tout si l'élément testé est l'ID.
- . Un combobox permettant de choisir la relation.
- . Un champ texte permettant de saisir la valeur de référence ; le type de la donnée saisie sera validé selon la nature de l'élément testé (ID → entier positif ; variable, formule → Number ; caactéristique → Number ou String selon le type de la caractéristique).
- . un bouton "Add" pour ajouter une nouvelle condition (vierge).△
- . un bouton "Mod" pour enregistrer les modifications apportées à la condition sélectionnée dans la liste au-dessus.△
- . un bouton "Sup" pour supprimer la condition sélectionnée dans la liste au-dessus.△

### 3.4 Onglet Conditional Style

**Définition :** Un affichage conditionnel (cf. partie 1.2) permet de définir un style d'affichage pour les boites vérifiant une certaine règle. L'ensemble des affichages conditionnels sont considérés en disjonction, mais dans l'ordre de leur priorité (entier positif de 1 à N s'il y a N affichages conditionnels définis), c'est à dire que toute boite utilisera le style du l'affichage conditionnel le plus prioritaire (plus petite valeur de priorité) dont elle satisfait la règle ; les boites ne satisfaisant aucune règle d'affichage conditionnel s'afficheront avec un style par défaut (paramétrable dans le menu Préférences). La règle d'un affichage conditionnel est identique à celle d'un filtre : c'est une conjonction de conditions portant sur n'importe quel élément d'une boite. Le style d'affichage des

boites se résume à leur couleur et à leur degré de transparence, les deux étant définis simultanément selon le codage RVBA.

Les affichages conditionnels sont gérés (créés, modifiés, supprimés) exclusivement depuis l'onglet Conditional Style. Ils reçoivent un identifiant (IDInt) unique à leur création et celui-ci n'est pas modifiable, à l'inverse de toutes leurs autres caractéristiques. La priorité par défaut d'un nouvel affichage conditionnel est d'être en dernière position ; à la suppression d'un affichage conditionnel, les suivants gagnent une priorité (compactage).

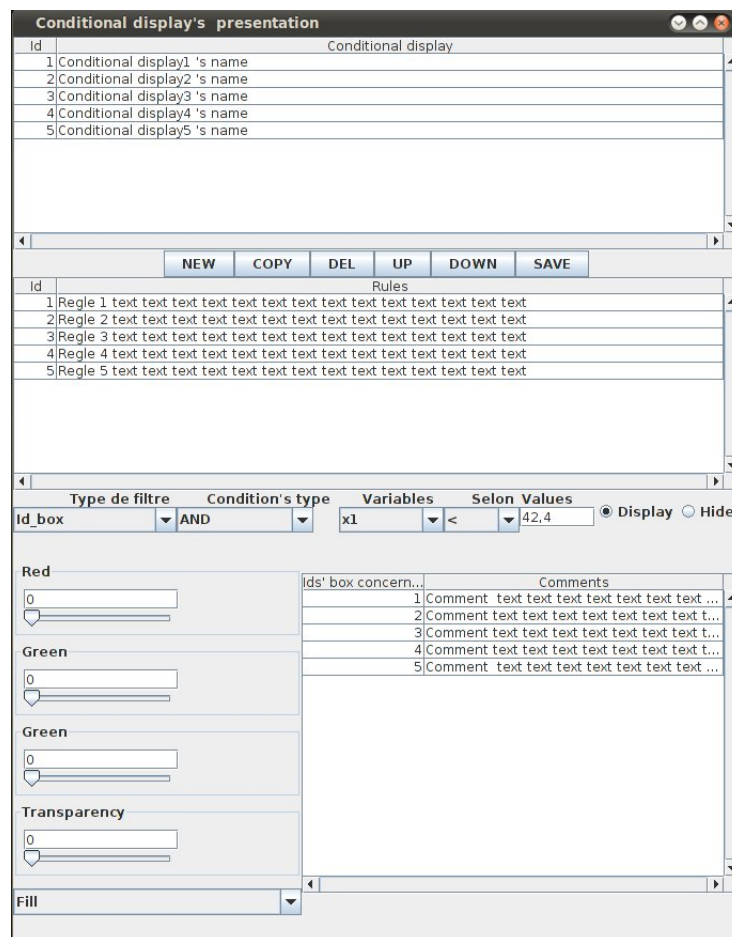


FIGURE 5 – Onglet de gestion des affichages conditionnels

#### Utilisation de l'onglet :

- Dans la partie supérieure, la liste des affichages conditionnels avec leur ID, leur label (sous forme de champ texte directement éditable dans la liste), leur priorité  $\Delta$  et le nombre de boites qui valident leurs règles  $\Delta$ . Cette liste pourra être réordonnée, de façon croissante ou décroissante, selon les ID, les labels, les priorités et le nombre de boites. Si la liste excède la taille de la zone dédiée, une barre de défilement verticale apparaîtra.
- Cinq boutons séparant la partie supérieure de la partie inférieure :
  - . New : Pour créer un nouvel affichage conditionnel vierge.
  - . Del : Pour supprimer l'affichage conditionnel sélectionné.
  - . Copy : Pour créer un nouvel affichage avec les même valeurs que celui sélectionné.
  - . Up : augmente la priorité de l'affichage conditionnel sélectionné.
  - . Down : diminue la priorité l'affichage conditionnel sélectionné.

- Dans la partie médiane, deux listes côte à côte :
  - . la liste des conditions constituant la règle de l’affichage conditionnel sélectionné (cf. onglet Filters).
  - . la liste des ID des boites validant la règle de l’affichage conditionnel sélectionné. La sélection d’une boite dans cette liste la sélectionne automatiquement dans l’onglet Box et la met en surbrillance dans la fenêtre de visualisation.
 Sous la liste des conditions se trouvent des champs éditables permettant d’ajouter, modifier et supprimer une condition (cf. Onglet Filters).△
- Dans la partie inférieure, la définition du style de l’affichage conditionnel sélectionné : 4 glissières associées à des champs textes permettant de personnaliser la couleur et la transparence selon le codage RVBA.  
 SUPPRIMER LE COMBO ET LA NOTION DE COULEUR DE TRAIT.  
 Un bouton Save permet d’enregistrer les modifications du style d’affichage.

## 4 Sauvegarde-Exportation

### Définition :

- Dans le cas d’une exportation textuelle, il s’agit simplement de la création d’un fichier de même nature que le fichier d’entrée avec en plus les données spécifiques à l’outil (cf 1.1) correspondantes au moment de la sauvegarde.
- Une exportation en image sera aussi possible :
  - Image bitmap (png, jpeg, ...)
  - vectorielle (svg, eps, ...)






# Image absente

FIGURE 6 – Fenêtre d’enregistrement

**Fonctionnement :** Une boîte de dialogue de sauvegarde classique, accessible depuis le menu File, par la barre d’outil, ou par le raccourci Ctrl+S, permettra de choisir le dossier en parcourant une arborescence. Une combobox permettra de choisir parmi les différents formats cités dans la définition. Un autre permettra de choisir si l’export concerne la totalité du pavage, seulement le pavage filtré, ou seulement la portion apparaissant dans le panneau de visualisation.

## 5 Visualisation

- Choix des une, deux ou trois variables à afficher. Cette sélection pourra s’opérer à l’aide de trois combobox présents dans la barre d’outils.
- Sélection : une entité (boite ou point) est sélectionnée par un clic du bouton gauche de la souris. L’objet sélectionné sera mis en surbrillance et sélectionné dans la liste de l’onglet correspondant.  
 ABANDONNER LA SELECTION MULTIPLE.
- Zoom :
  - . revenir à une vue globale homogène du pavage (filtré). Soit par une touche ( **F10** ), soit par un bouton de la barre d’outil, soit par le menu «display» : «Zoom All».
  - . zoomer/dézoomer (homogène) soit avec la molette de la souris dans la zone de visualisation (zoom centré sur la zone pointée sur la souris), soit par les touches **+** et **-** (clavier ou pavé numérique), soit par le menu «display» : «Zoom In/Out», soit en donnant directement le pourcentage de zoom (par rapport à la vue globale = 100%) dans la barre d’outil, soit enfin par le menu contextuel.

- . zoom par dimension : idem zoom/dézoom homogène en maintenant la touche  ,  ou  (clavier ou pavé numérique) enfoncée.
- Déplacement :
  - . Transposition 2D, soit avec les flèches du clavier, soit à la souris en maintenant le bouton gauche enfoncé.
  - . Centrage sur le point de mire de la souris, soit grâce à la touche  soit via le menu contextuel.
  - . Définition textuel des range possible par le menu «display» : «ranges»
- Rotation (en visu 3D uniquement) : s'effectue en maintenant enfoncée la touche  ou la molette de la souris et déplaçant la souris ou avec les flèches du clavier.

# Table des figures

2.1	Cas de la fonction cosinus . . . . .	7
2.2	Comparaison graphique entre la Hull-consistance (rouge) et la 3B-consistance (orange) . . . . .	8
2.3	Exemples d'application dans la robotique . . . . .	9
2.4	Intersection de deux disques . . . . .	12
3.1	arbre binaire . . . . .	17
3.2	a-b arbre . . . . .	17
3.3	Représentation d'un QuadTree où les données sont des points . . . . .	18
3.4	Superposition de boîtes . . . . .	19
3.5	État initial du QuadTree lorsque deux boîtes sont côte à côte . . . . .	20
3.6	État final du QuadTree lorsque deux boîtes sont côte à côte . . . . .	20
3.7	Représentation d'un R-tree[Wika] . . . . .	21

# Bibliographie

- [A.GUTTMAN84] A.GUTTMAN. R-Trees : A Dynamic Index Structure for Spatial Searching. <http://www-db.deis.unibo.it/courses/SI-LS/papers/Gut84.pdf>, 1984.
- [cho99] Librairie CHOCO. <http://www.emn.fr/z-info/choco-solver/>, 1999.
- [C.JERMANN02] C.JERMANN. *Résolution de contraintes géométriques par rigidification récursive et propagation d'intervalles*. PhD thesis, Université de Nice-SOPHIA ANTIPOLIS UFR SCIENCES, Decembre 2002.
- [F.GOUALARD00] F.GOUALARD. *Langage et environnements en programmation par contraintes d'intervalles*. PhD thesis, Université de Nantes, Juillet 2000.
- [gnu] Gnuplot Web Page. <http://www.gnuplot.info/>.
- [H.SCHICHL03] H.SCHICHL. *Mathematical Modeling and Global Optimization*. PhD thesis, University of Vienna, November 2003.
- [LF06] L.GRANVILLIERS and F.BENHAMOU. Algorithm 852 : Realpaver : an Interval Solver using Constraint Satisfaction Techniques. [http://pagesperso.lina.univ-nantes.fr/~granvilliers-l/papers/toms\\_2006.pdf](http://pagesperso.lina.univ-nantes.fr/~granvilliers-l/papers/toms_2006.pdf), 2006.
- [L.GRANVILLIERS04] L.GRANVILLIERS. Realpaver manual. <http://pagesperso.lina.univ-nantes.fr/~granvilliers-l/realpaver/src/realpaver-0.4.pdf>, Août 2004.
- [lin] Laboratoire LINA. <http://www.lina.univ-nantes.fr/>.
- [opt] Équipe OPTI . <http://www.lina.univ-nantes.fr/?-OPTI-.html>.
- [ort11] Google or-tools. <http://code.google.com/p/or-tools/>, 2011.
- [Wika] Wikipedia. R-tree. <http://en.wikipedia.org/wiki/R-tree>.
- [Wikb] Wikipédia. Conditionnement d'une matrice. [http://fr.wikipedia.org/wiki/Conditionnement\\_\(analyse\\_numérique\)](http://fr.wikipedia.org/wiki/Conditionnement_(analyse_numérique)).
- [YAAY06] Y.MANOLOPOULOS, A.NANOPOULOS, AN.PAPADOPOULOS, and Y.THEODORIDIS. *R-Trees : Theory and Applications*. Springer, 2006.