Institut Supérieur
d'Informatique de Modélisation et
de leurs Applications

Campus des Cézeaux
24 avenue des Landais
BP 10 125
63 173 AUBIERE CEDEX

Laboratoire de
Physique Corpusculaire

Campus des Cézeaux
24 avenue des Landais
BP 80 026
63 173 AUBIERE CEDEX

Rapport d'ingénieur

Projet de 2e année
Filière Calcul et Modélisation Scientifique

Rapport de Projet

Étudiant :
Josselin Massot

Tuteur :
Dr Bogdan Vulpescu
Tuteur ISIMA :
Monsieur Isima

Projet de 60h mars 2015

Remerciements

Je tiens avant tout à remercier mon tuteur, Bogdan Vulpescu, ingénieur de recherche au LPC-Clermont. Il m'a laissé une grande liberté quant aux possibilités de développement, encadré et soutenu dans le calme.

Je voudrais remercier Alain Falvard, directeur du LPC pour son accueil au sein de la structure qu'est le LPC, mais aussi à toute l'équipe du service informatique qui m'a très rapidement intégré parmi eux.

Mes remerciments vont aussi à Alexandre Claude pour son soutient, les infatigables débats que nous avons partagés, ainsi qu'Emmanuel Delage pour la place que tous les deux m'ont laissée dans leur bureau.

Je souhaite remercier plus particulièrement, Emmanuel Gangler, chargé de recherche au CNRS, sans qui ce stage n'aurait pas eu lieu.

Liste des figures, tableaux, algorithmes et extraits de code

Liste des figures

2.1 Logo de LSST 5
4.1 *Benchmark* entre *C++* et *Python* 8
8.1 Exemple de partitionnement et d'arbre associé 14
8.2 Espace de travail à 2 ou 3 dimensions 15
8.3 Relation entre la géométrie de l'ensemble et le *quadtree* généré 17
8.4 Relation entre la géométrie de l'ensemble et le *kd-tree* généré 18

Liste des tableaux

Liste des algorithmes

7.1 Algorithme de recherche directe 13
8.1 Algorithme récursif de construction d'un arbre de partitionnement 16

Liste des extraits de code

Résumé – Abstract

Résumé

Ceci est mon résumé en français
sur plusieurs lignes

Mots clés : clé, clé, clé, clé

Abstract

This is my abstract in english with some lines
and an other line here

Keywords: key, key, key, key

Table des matières

Remerciements i

Liste des figures, tableaux, algorithmes et extraits de code ii

Résumé – Abstract iii

Table des matières v

Glossaire vi

1 Introduction 1

I Introduction de l'étude 3

2 Présentation du cadre d'étude 4
2.1 Le projet LSST 4
2.2 Le *Stack* et `DC-2013` 5
2.2.1 La *Data-Challange Summer 2013* 5
2.2.2 Conclusion de la `DC-2013` 6

3 Sujet d'étude 6
3.1 Stockage et accès aux données 6
3.2 Comparaison des données 7

4 Outils et prévisions 7
4.1 Outils de développement 7
4.2 Outils de gestion de projet 9

II Méthodes et résolution 10

5 Organisation des données 11
5.1 Détails des données 11
5.1.1 Définition d'un objet et d'une source 11
5.1.2 Impact sur les données 11
5.2 Structure de la base 11
5.2.1 Modèle en étoile 12

5.2.2 Modèle à colonnes variables 12

6 Jeu d'essai **12**
6.1 Objectif 13
6.2 Contrainte du jeu 13

7 Recherche directe **13**

8 Arbres de partitionnement **14**
8.1 Définition et propriété de l'espace de données 15
8.2 Algorithme de construction 16
8.3 Choix de la division 16
8.3.1 *quad-tree* 16
8.3.2 *kd-tree* 17

9 Dérécursification **18**
9.1 Objectif 19

III Résultats et discussion **20**

10 Base de données **21**
10.1 Le retour client 21
10.2 Efficacité 21
10.3 Maintenabilité 21

11 Algorithmes de comparaison **21**
11.1 Efficacité 21
11.1.1 Temps d'exécution 21
11.1.2 Utilisation mémoire 21
11.2 Crédibilité du programme 21

12 Analyse des résultats **21**
12.1 Résultat de la comparaison 21
12.2 Conclusion sur le *Stack* 21

13 Conclusion **22**

Bibliographie & Webographie **a**

Glossaire

Complexité algorithmique : Complexité algorithmique c'est un calcul formel sur le nombre de ressources maximales (temps ou espace) nécessaire pour exécuter un algorithme. Elle dépend de la quantité de données, notée n.

Coordonnées astronomiques : Coordonnées astronomiques les coordonnées astronomiques les plus souvent utilisées sont l'ascension droite et la declinaison, abrégées en RA (*right ascension*) et Dec (*declinaison*). Ce sont les projections respectives de la longitude et la lattitude terrestre sur le ciel. La zone observée ici est proche de l'équateur céleste, donc 0° de déclinaison.

LSST : LSST (*Large Synoptic Survey Telescope*) est un projet de télescope grand champ qui sera construit sur le site de Cerro Pachón, au nord du Chili, à 2680 mètres d'altitude. Les premières images disponibles en 2020 devraient permettre d'approfondir nos connaissances de l'Univers par la détermination de la nature de l'énergie noire et de la matière noire, un inventaire du système solaire et une cartographie de la Voie Lactée. Le LPC est impliqué à plusieurs niveaux dans ce projet.

SDSS : SDSS (*Sloan Digital Sky Survey*) est un télescope de 2,5 m dédié à la surveillance du ciel situé au Nouveau-Mexique (États-Unis) dont les observations se sont étalées de 2000 à 2007.

***Stack* :** *Stack* Logiciel de traitement d'images du télescope LSST, cela fait aussi référence à la suite de bibliothèques et de logiciels faisant fonctionner ce logiciel. Au long de ce stage je n'ai pas eu l'occasion d'ajouter ou de modifier des modules déjà existant, le terme *Stack* fera donc toujours référence au logiciel.

1 Introduction

L'utilisation de l'outil informatique pour réaliser des analyses longues ou poussées de données, ainsi que des simulations complexes est devenue courante dans le milieu scientifique. Ces études produise une quantité importante de données qu'il est nécessaire de stocker et d'évaluer leur qualité.

Le problème de stockage est souvent associé à ce que l'on appelle le *Big Data*, littéralement *grandes données*. Il regroupe tous les problèmes liés au stockage de grande quantité de données, à la variété, c'est-à-dire le stockage de données complexes (*e.g.* images, vidéos, objets sérialisés) et à la vélocité de la base de données, ce qui signifie la garantit de l'accès à ces données. Nous ne parlerons plus de *Big Data* par la suite puisque nous nous limiterons à l'utilisation de *systèmes de gestion de base de données* (SGBD) dit *classiques*, sous-entendu ceux utilisés avant l'apparitions de SGBD spécialisé dans le *Big Data* comme SciDB ou Qserv.

Le problème de la qualité des données est lié à la comparaison à des données déjà existantes dans le même domaine. Cette analyse n'est pas toujours évidente car l'existance de ces dernières n'est pas assurée, ce qui peut être contraint par l'aspect novateur de certaines expériences. Dans notre cas nous possédons une base de données de références, dans certains cas il est nécessaire de générer des données simulées à l'aide du modèle théorique de l'expérience. La comparaison de deux bases de données s'effectuent pas des associatioins entre éléments de la première base avec des éléments de la seconde ; pour cela on utilise les algorithmes de recherche de plus proche voisin, de cette manière nous pouvons associer un élément avec l'élément le plus proche de l'autre base.

Ce projet a pout but d'évaluer la qualité du logiciel de traitement d'images du futur télescope LSST : le *Stack*[(1)], mais aussi de stocker les données utiles à cette évaluation et le résultat de cette évaluation dans une base de données.

Le télescope LSST est prévu pour la surveillance du ciel profond et des objets très faibles de notre système solaire, c'est-à-dire qu'il ne s'intéresse pas aux grands astres du système solaire, mais à des objets plus lointains tels que d'autres galaxies, des nébuleuses ou des étoiles il s'intéresse aussi à des objets plus proches comme des astéroïdes. La surveillance du ciel permet de repérer des évènements astronomiques de courtes durées comme des supernovæ qui sont considérées comme des "chandelles astronomiques"

(1). Le terme *stack* signifiant en anglais "pile" nous utiliserons aussi bien le féminin (pour rappeler la traduction du mot) que le masculin (puisqu'il s'agit d'un terme anglais sans genre, le masculin prend généralement le pas).

permettant des calculs de distances d'objets. L'astrométrie, c'est à dire la mesure des coordonnées des étoiles doit être la plus précise possible pour en observer la déviation dans le temps.

Le logiciel *Stack* a pu être testé l'été dernier au cours de la *Data-Challange 2013* (`DC-2013`) sur les données du télescope SDSS, sur la *stripe 82* qui est une bande du ciel proche de l'équateur céleste. L'objectif de ce projet est donc d'évaluer la qualité des données générées par la `DC-2013`, ainsi que mesurer l'erreur commise par le *Stack* par rapport aux mesures de SDSS. Deux analyses seront effectuées, une première de comparaison à une autre base de données, en l'ocurence ici la base de données de SDSS, la seconde d'étude de stabilité temporelle de la base.

Première partie

Introduction de l'étude

2 Présentation du cadre d'étude

2.1 Le projet LSST

Le télescope LSST (*Large Synoptic Survey Telescope*) est un projet mondial pour construire un télescope de surveillance continue du ciel. Il fait suite à des télescopes tels que SDSS ou Vista [2] qui ont pour but de rechercher des phénomènes astronomiques brusques et non prévisibles comme les supernovæ.

Le projet LSST est piloté par le laboratoire américain SLAC [3]. Depuis son entrée dans le consortium en 2007, la collaboration LSST France compte aujourd'hui 8 laboratoires du CNRS provenant du département de recherche IN2P3 (Institut National de Physique Nucléaire et Physique des Particules). Ces laboratoires sont, par ordre alphabétique :

- **APC** (AstroParticules et Cosmologie) (Paris), pour la calibration et le contrôle commande de la caméra et le calcul ;
- **CC-IN2P3** (Centre de Calcul IN2P3) (Lyon), calcul et gestion des données LSST ;
- **CPPM** (Centre de Physique des Particules de Marseille) (Marseille), pour le changeur de filtres et le calcul ;
- **LAL** (Laboratoire de l'Accélérateur Linéaire) (Orsay), pour l'électronique des CCD [4] ;
- **LMA** (Laboratoire des Matériaux Avancés) (Villeurbanne), pour mener la phase d'étude de faisabilité des filtres LSST ;
- **LPC** (Laboratoire de Physique Corpusculaire) (Clermont-Ferrand), pour le banc de test du système d'échange de filtres , et le calcul ;
- **LPNHE** (Laboratoire de Physique Nucléaire et de Hautes Énergies) (Paris), pour le carrousel de filtres, le banc de caractérisation de la caméra, l'électronique des CCD et le *firmware* associé à l'électronique de contrôle et de lecture des CCD .
- **LPSC** (Laboratoire de physique subatomique et de cosmologie) (Grenoble), pour le banc de caractérisation de la caméra et le chargeur de filtres.

Trois grands sujets d'étude ressortent de cette liste, la partie *caméra* ayant rapport à la construction, aux tests et au développement de l'application de contrôle de la caméra, la partie dite *calcul* ayant rapport aux développements d'applications, au traitement et

(2). Vista : (*Visible and Infrared Survey Telescope for Astronomy*) télescope de l'ESO (European Southern Observatory) au Chili à partir de 2007.

(3). SLAC : (*Stanford Linear Accelerator Center*) Centre de l'accélérateur linéaire de Stanford, laboratoire de physique situé en Californie (États-Unis).

(4). CCD : (*Charge-Coupled Device*) il s'agit d'un capteur photométrique permettant la prise d'images astronomiques.

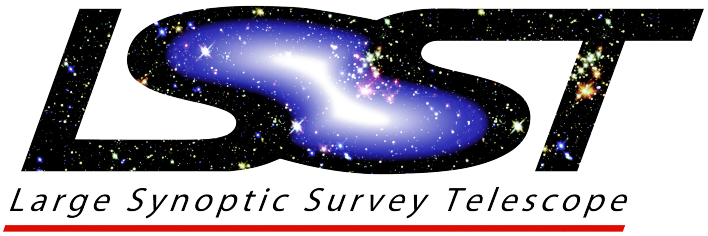


Figure 2.1 – Logo de LSST.

stockage des données ainsi que le logiciel de calibration de la caméra, et finalement une partie dite *science* s'occupant de l'interprétation des résultats.

2.2 Le *Stack* et DC-2013

Le *Stack* est le logiciel de traitement d'images du projet LSST. Contrairement à d'autres programmes jouant le même rôle sur d'autres télescope, celui-ci est pourvu d'un module de calibration le permettant de fonctionner avec des images en provenance de n'importe quel appareil. En effet ce module permet de transformer une image en une image calibré qui aura les mêmes propriété qu'importe sa provenance.

Le logiciel *Stack* est encore en cours de développement. Pour des raisons de gestion du projet, seul le personnel du laboratoire du SLAC est autorisé à participer à son développement, par conséquent très peu d'indications concernant la méthode de traitement d'image et le repérage des sources lumineuses sont disponibles.

2.2.1 La *Data-Challange Summer 2013*

Durant l'été 2013, un test du *Stack* sur une partie des données de SDSS a été effectué en parallèle au centre de calcul de l'IN2P3 à Lyon (CC) et aux États-Unis. Pour cela les données brutes de SDSS de la *stripe 82* ont été récupérées, puis traitées par le logiciel *Stack*.

Pour la partie française, 300 nœuds de calculs ont été mobilisés au CC de juin à octobre, pour analyser plus de 5 To de données. Le temps total de calcul est d'environ 300 000 heures, ce qui représente 34 ans de calculs. On voit déjà l'importance de la parallélisation des calculs qui permet d'effectuer plusieurs calculs simultanément sur plusieurs nœuds.

La base de données générées est au format MySQL, fait 4,3 To (770 Go d'index (18%) et 3,6 To de données (82%), c'est environ 2 milliards de lignes par table, sachant qu'il y

a 5 tables. L'indexation des données prend à elle seule 15 heures de calculs par table.

Cette expérience fut mené aussi aux États-Unis. Le nom donné à l'évènement fut *Data-Challange Summer 2013* aussi abrégé en `DC-2013`.

2.2.2 Conclusion de la `DC-2013`

Aucune analyse de la qualité des résultats ne fut effectuée en dehors d'un stage l'année passé, l'expérience reste donc plus une étude de faisabilité ainsi qu'un exploit technique. Il est maintenant nécessaire d'étudier les résultats du *Stack*, ceci permet de connaître mieux le logiciel ainsi que la qualité de ses résultats. Le stage que j'ai pu effectué l'an passé a permi une première analyse des résultats à l'aide d'un seul algorithme, mais les données nécessaire à ce traitement et les résultats obtenus sont stocker sous forme de fichiers texte.

Les calculs effectués aux États-Unis ainsi qu'en France n'ont pas porté sur le même jeu de données, seule une petite partie était commune pour vérifier que les résultats était semblable qu'importe la longitude. Le D^{r} Phillipe Gris du LPC a effectuer cette comparaison de résultats, comme attendu la différences entre les résultats du *Stack* de part et d'autre de l'Atlantique sont nuls.

3 Sujet d'étude

Ce projet d'année est la suite d'un stage déjà effectué au LPC, ainsi le travail de préparation a déjà été effectué à cette occasion. L'aspcet théorique de certains algorithmes ont déjà été abordé, un modèle de base de donnée fut aussi étudié sur le papier. Ce projet est l'aboutissement de ce travail préparatoire, il s'agit donc d'approfondir les détails théoriques, d'implémenter ces algorithmes et un grand travail de test essentiellement sur la base de données ainsi que sur les différents algoritmes d'analyse. Ce travail de test permettra de déterminer la qualité des programmes d'analye de qualité pour en déduire lequel s'approprie le mieux à l'analyse des résultats du *Stack*.

3.1 Stockage et accès aux données

Les données nécéssaires à l'analyse de qualité ne sont pas accessible facilement, ainsi un problème rencontré au cours du stage fut l'accès aux données du télescope SDSS. En effet une limitation de 10 requêtes par minutes est imposé par la base de données de

SDSS, cela leur permet de garantir un accès public à tous, mais contraint l'exploitation en masse de résultats.

Les données générés par le *Stack* sont situés dans une multitude de fichiers au format `fits`. Le temps de lecture de ces fichiers est relativement lent surtout pour un traitement en masse de toute la portion de la *stripe 82* analysée par le CC-IN2P3.

Pour résoudre ces problèmes d'accès aux données amonts à l'analyse, il fut décidé pendant le stage de stocker ces données dans des fichiers textes. Ce choix est en parti dû à des contraintes techniques d'accès à une base de données ainsi que de temps de conception de celle-ci. Les résultats de l'analyse furent aussi stockés dans des fichiers textes au format `csv`. Ce format permet très simplement de stocker des données tabulaires.

Le problème que nous rencontrons avec ces fichiers texte est l'accès lent et difficile aux données. En effet nous ne disposons pas de la puissance d'une base de données relationnelles qui lié au langage SQL permet d'effectuer des recherches selon n'importe quel paramètres de manière aisé et rapide. De plus il est impératif d'avoir un produit robuste à la monté en charge, en effet seul 1% des données totales furent analysé pendant le stage, et cela représenté déjà 2 millions de sources, donc de lignes, à sauvegardé. Ainsi le choix fut pris par l'équipe du LPC de stocker ces données dans uen base de données.

3.2 Comparaison des données

La comparaison des résultats du *Stack* lors de la `DC-2013` et ceux de SDSS s'effectue par association. En effet à chaque élément découvert par SDSS on essaie d'y associer un élement traité par le *Stack*. L'association s'effectue selon des critères à la fois astrométriques (les coordonnées de nos sources lumineuses), mais aussi photométrique (la luminosité de nos sources lumineuses). Par conséquent cela correspond à de la recherche de ressemblance ou en tout cas de la recherche de plus proche voisin.

La recherche de plus proche voisin est un sujet vaste qui s'applique aussi bien à des domaines comme

4 Outils et prévisions

4.1 Outils de développement

Ce projet vivant au milieu du projet LSST, il est construit sur les mêmes bases. Le projet LSST impose l'utilisation du couple *C++/Python*.

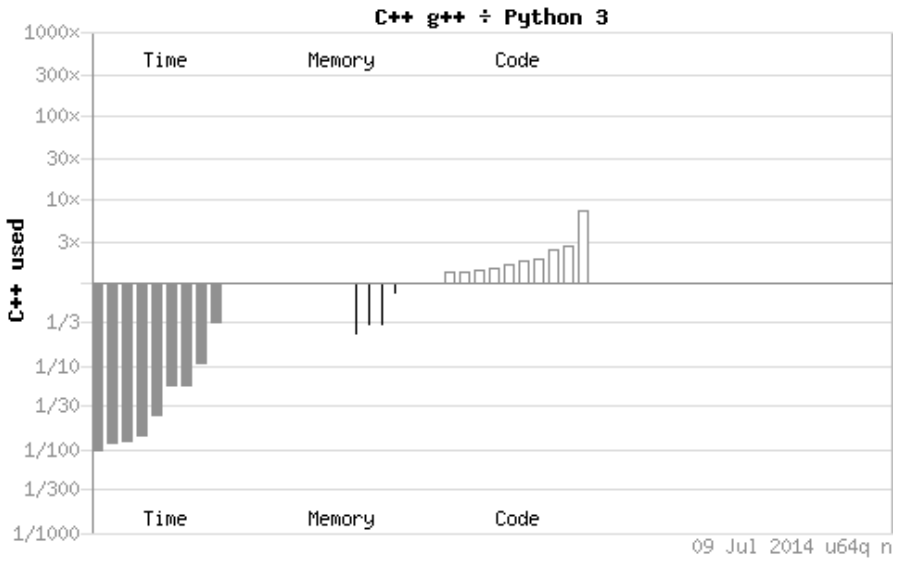


Figure 4.1 – Étude de benchmark *entre* C++ *et* Python. *Cette étude compare les performances de* Python *par rapport au* C++ *en effectuant le quotient* $^{Performance\ C++}/_{Performance\ Python}$*, donc les résultats en dessous de 1 sont en faveur de* C++*, ceux au dessus de 1 en faveur de* Python. *Les tests s'effectuent en 3 catégories, temps d'exécution, utilisation de la mémoire et nombre de lignes de code. Chaque barre de l'histogramme dans une catégorie correspond à un algorithme (*n-body*,* mendelbrot*,* binary-trees*, etc.). Source :* `http://benchmarksgame.alioth.debian.org/`

C++ Le langage *C++* est un langage compilé, ce qui signifie que le code est d'abord converti en langage machine au cours de la compilation pour former un exécutable propre à une architecture (matériel et logiciel). L'intérêt réside dans la vitesse d'exécution, puisque l'exécutable est directement lu par la machine sans le moindre interpréteur.

Le *C++* est un langage relativement bas niveau, cela signifie que la gestion de la mémoire est en bonne partie laissée au programmeur. Cela rend le développement plus long, et nécessite un certain nombre de connaissances pour éviter des erreurs de programmation.

Du fait de sa vitesse d'exécution le *C++* est souvent utilisé pour des moteurs de calcul, ou des tâches sur un grand nombre de données.

Python Le langage *Python* est un langage interprété, ce qui signifie qu'au moment de l'exécution, chaque ligne est traduite en langage machine avant d'être exécutée, et ce à chaque exécution d'un même script *Python*. Les performances du *Python* sont jusqu'à 100× moindre que celles du *C++* comme le montre la figure 4.1. *Python* est certes moins performant que *C++*, mais il a l'avantage de facilité la programmation avec un nombre

de lignes de code moins élevé, des fonctions haut niveau dans la bibliothèque standard et l'interpréteur qui permet un débogage plus rapide. *Python* est donc privilégié pour la partie interface ainsi que l'ordonnancement des différentes tâches, laissant le cœur du calcul à *C++*.

MySQL La partie base de données implique le choix d'un système de gestion de base de données (SGBD), celui-ci est MySQL. Ce choix est imposé par l'installation de ce logiciel dans l'environnement de développement qu'est le CC-IN2P3.

4.2 Outils de gestion de projet

Méthode kanban en ligne (chercher ou développer un outils pour, et proposer un accès à Bogdan).

Diagramme de Gantt (pas le choix y'a Mouzat), permet planifier des tâches avec des "deadlines", d'affecter une tâche à une équipe, mais ici je suis seul à travailler sur le projet, un diagramme prévisionnel est disponible ici mais il est fortement hypothétique car je suis incapable de prévoir certains inprévu technique dû à mon statut d'externe au LPC. Je suis obligé de passer par certaines machines etc.

Deuxième partie

Méthodes et résolution

5 Organisation des données

5.1 Détails des données

Nos données sont les résultats du *Stack* lors de la `DC-2013`, ainsi que les données de SDSS de la *stripe 82* accessible depuis le Web. Ces données sont les résultats d'un traitement d'images captées par le télescope SDSS, il s'agit de sources lumineuses.

5.1.1 Définition d'un objet et d'une source

L'objet fait référence à l'astre que l'on observe dans le ciel dont données astrophysiques sont connus. Celles-ci inclus les coordonnées, la magnitude, la variation de la magnitude, ainsi que la parallaxe qui correspond au changement de coordonnées dû à la rotation de la Terre autour du Soleil.

La source quant à elle ne correspond qu'à l'image à un instant t d'un objet. Les sources sont les éléments listés par le *Stack*, de même la base de données de SDSS est un catalogue de sources et non d'objets.

5.1.2 Impact sur les données

Les sources que nous analysons ont été observés à travers différents filtres allant de l'infra-rouge à l'ultra-violet, pour un total de 5 filtres. L'objet astrophysique est représenté à un instant t par une source dans chaque filtre, l'association des sources entre les filtres a déjà été faite dans la base de données de SDSS, ce n'est pas le cas du *Stack*. Ainsi dans les données de SDSS nous retrouvons un calcul de magnitude apparente dans tous les filtres pour une source ; dans les données du *Stack* il y a une source différente par filtre, il y a donc un jeu de coordonnées par filtre.

5.2 Structure de la base

Deux modèles relationnels de la structure de la base de données ont été pensé. Le premier est un modèle classique et fiable, il s'agit du modèle dit « en étoile ». Le second est un modèle pensé pour optimiser le temps d'obtention des données de l'association, mais touche, à chaque ajout d'algorithme d'association, au modèle relationnel.

La base de données que nous souhaitons construire doit stocker à la fois les données nécessaire à l'analyse que nous allons effectuée (informations sur les sources du *Stack* et de SDSS), des données sur les différents algorithmes que nous allons utiliser ainsi que les

résultats de ces algorithmes. Ces résultats étant l'association d'une source du *Stack* avec une source de SDSS.

5.2.1 Modèle en étoile

Il s'agit d'un modèle avec 4 tables. Tout d'abord deux tables pour stocker nos sources, une table pour les sources du *Stack* et une pour les sources de SDSS. Ces tables sont crées et fixées lors de la récupération des données.

Une table est réservée pour les informations relatives aux algorithmes et contiendra des données comme le nom de l'algorithme, l'auteur, ainsi que la méthode utilisée par celui-ci, dans la pratique seul l'identifiant de l'algorithme nous intéressera.

Enfin les données relatives aux résultats de ces algorithmes sont tous stockés dans une même table. Ces données sont associées à un identifiant unique, et contiennent l'identifiant de la sources du *Stack* que l'on associe et celui de la source de SDSS associée, mais aussi l'identifiant de l'algorithme utilisé.

Les deux tables contenant les sources ont une taille similaire n, par conséquent la table contenant les résultats de l'association contiendra n données par nouvel algorithme si les algorithmes utilisés n'associent une source qu'à une seule source. Si un algorithme associe une source à plusieurs sources, ces données sont tout simplement stockées à l'aide de plusieurs identifiant unique dans notre base.

5.2.2 Modèle à colonnes variables

Il s'agit d'un modèle avec 3 tables. Comme précédement deux tables sont dédiées au stockage des sources, ainsi qu'une table pour les informations des algorithmes. Mais contrairement au modèle en étoile, on agit sur les deux tables contenant les sources en ajoutant une colonne par algorithme et en y stockant l'identifiant de la source associée. Ceci permet d'obtenir directement la liste des sources de l'autre base associées à une source.

Le problème est la modification du modèle relationnel qui peut entraîner une perte des données ou des lenteurs d'accès dû à l'ajout de données sur chaque ligne de la base.

6 Jeu d'essai

6.1 Objectif

Pouvoir tester les algorithmes d'association, explication du tri en vrai position, faux positif, vrai négatif et faux négatif.

6.2 Contrainte du jeu

Il faut que le jeu d'essai soit proche de ce que l'on veut tester, définir un écrat moyen, un écart maximal (loi normal ?), l'ajout d'erreur ?

Au moins une des base est généré aléatoirement, donc limitation par le générateur de nombre pseudo aléatoire qui peut ou non faire apparaître des comportements.

7 Recherche directe

La première approche utilisée pour rechercher le plus proche voisin est une recherche dite diecte qui consiste à pour chaque point de la première base de données, on parcourt la seconde base, puis on compare le point de la première base et le point courant de la seconde.

Algorithme 7.1 Algorithme de recherche directe

```
1: d_min ← 100 ;        ▷ Valeur arbitrairement grande de distance minimal d'initialisation
2: Pour point_i dans la première base Faire
3:     Pour point_j dans la deuxième base Faire
4:         Si d(point_i, point_j) < d_min Alors        ▷ d fonction de calcul de distance
5:             d_min ← d(point_i, point_j) ;
```

Cet algoritme a une complexité en $\mathcal{O}(n^2)$, il s'agit donc d'un algorithme relativement lent pour le problème à traiter, il n'est donc pas intéressant de l'utiliser sur un grand nombre de données. Malgré cela, cet algorithme est utilisé par sa modularité, en effet il est très simple de modifier la condition à la ligne 4 pour y prendre en compte la liste detous les paramètres que nous souhaitons prendre en compte.

L'optimisaton d'un tel algorithme peut se faire dans des cas particulier en effectuant un pré-traiement, d'une complexité parfois plus importante, permettant ensuite de réaliser un grand nombre de calcul rapidement. Il est donc nécessaire de trouver un juste milieu entre le temps d'éxécution du pré-traitement, et le gain obtenu par la suite au moment du traitement.

Il est aussi possible de lancer cette algorithme non pas sur toutes les données, mais sur un nombre restreint de données en découpant l'esapce en plusieurs morceaux. Cela équivaut à effectuer un partitionnement de l'espace.

8 Structure d'arbres de partitionnement

Certains algorithmes, ou approches algorithmiques, nécessite un partitionnement de l'espace des données pour diviser le travail et n'effectuer que des tâches élémentaires. Le pincipe de partition est proche d'une vision récursive, en effet on divise l'espace de travail tant que l'on ne se retrouve pas dans un cas simple que l'on sait traiter. Ceci permet de chercher le plus proche voisin mais aussi de créer des régions d'intéraction, cette dernière problématique ne nous intéressera pas ici. Le partitionnement est ensuite représenté par la structure d'un arbre comme une arborecense du partitionnement.

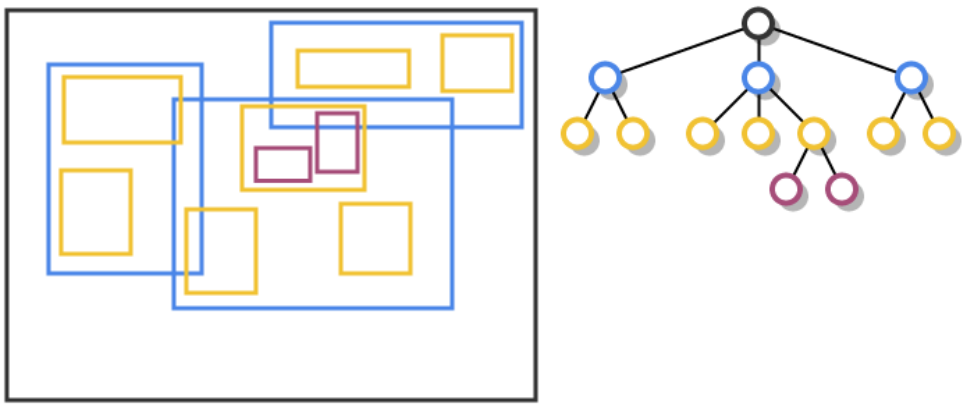


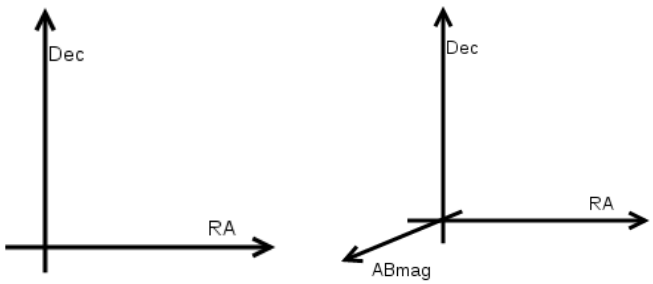
*Figure 8.1 – Exemple d'arbre de partitionnement et de l'abre qui lui est assicié. Le principale intérêt est de travailler sur des ensembles de cardinal moins élevé, ou de mettre en valeur des espaces d'intéraction (*R-tree *par exemple).*

Deux structures d'abre de partitionnement seront étudiées ici pour recherhcer le plus proche voisin, il s'agit du *kd-tree* et du *quad-tree*. Nous verrons les avantages et les inconvéniants des deux. Chaque nœud de l'arbre représente une partition de l'espace, ses fils ses sous-partitions. Plusieurs partitionnements sont envisageables pour obtenir des divisons avec des propriétés plus ou moins intéressantes.

8.1 Définition et propriété de l'espace de données

Avant de parler de l'algorithmie et des structures de données et il est important de parler des espaces dans lesquels nous allons travailler et de ce que nous voulons y faire. Les données que nous allons traiter sont des points dans un espace à 2 dimensions, et nous souhaitons rechercher le plus proche voisin. La définition du plus proche voisin n'est ici pas seulement d'ordre géométrique, en effet il est possible d'ajouter d'autres paramètres. La méthode choisi ici est dans un premier temps d'effectuer une discimination géométrique, puis le dernier choix selon d'autres critères.

Il est aussi possible de travailler en 3 dimensions, notre objectif est de travailler sur des sources lumineuses dans le ciel, par conséquent nous nous retrouvons avec deux dimensions d'espace (l'ascension droite et la déclinaison), mais il est aussi possible d'ajouter la magnitude (luminosité) comme dimension. Ainsi nous travaillerions dans un espace à 3 dimensions, dont 2 dimensions d'espace. Cette technique ne sera pas exploité ici car cela donne autant d'importance aux paramètres astrophysiques (ascension droite et déclinaison) qu'à un paramètre photométrique (magnitude) qui est fortement soumis aux erreurs de calibration et des conditions de mesure. Mathématiquement l'ajout de dimensions non spatiales (le temps ou un paramètre) n'est pas dérangeant et n'influe en rien la théorie. Ainsi il n'est pas rare de travailler en n dimensions en fixant n selon le besoin.



(a) Espace à 2 dimensions. (b) Espace à 3 dimensions.

Figure 8.2 – L'espace de travail peut être aussi bien à 2 qu'à 3 dimensions. L'exemple 8.2a montre un espace à 2 dimensions d'espaces ; l'exemple 8.2b montre un exemple d'espace à 2 dimensions d'espace et une de paramètres.

La généralisation à k dimensions est possible mais elle ne nous intéressera pas mais les différences des structures seront évoquées pour éveiller la curiosité du lecteur.

8.2 Algorithme de construction

L'algorithme de construction de la structure du *kd-tree* et du *quad-tree* sont similaires, seule l'étape de division différe et est propre à chacune des structures. Le choix de la division implique des propriétés différentes sur les arbres générés. Cette étape sera détaillée en même temps que les différences des deux structures.

L'algorithme de construction est, sous sa forme la plus simple, récursif. Celui-ci est décrit dans l'algoritme 8.1.

Algorithme 8.1 Algorithme récursif de construction d'un arbre de partitionnement

```
1: Fonction CONSTRUIRE_ARBRE(espace)
2:     Si nombre de points de l'espace = 0 Alors ▷ L'espace ne contient pas de source, impossible de diviser
3:         Retourner espace ;
4:     Sinon Si nombre de point dans l'espace = 1 Alors ▷ L'espace ne contient plus qu'un point, il n'est plus nécessaire de diviser
5:         Retourner espace ;
6:     Sinon ▷ Il faut diviser l'espace, et construire les sous espaces
7:         Retourner CONSTRUIRE_ARBRE( DIVISER_ESPACE(espace) ) ;
```

Le principe est simple, tant que l'espace contient plus d'une source on construit l'arbre sur les sub-divisions de l'espace. Quand il n'y a plus qu'une ou zéro source, on stope la division.

8.3 Choix de la division

La différence entre la structure du *quad-tree* et du *kd-tree* est la méthode de divison de l'espace. Le partitionnement choisi possède des propriétés dépendante de cette méthode.

8.3.1 *quad-tree*

La division du *quad-tree* consiste à sub-diviser l'espace en quatre parties en prennant le milieu de chaque côté. Ceci permet de construire un arbre dont chaque nœud possède 0 ou 4 fils.

La construction de cette arbre présente l'avantage d'être purement géométrique, ainsi il est possible d'effectuer un premier partitionnement pour utiliser un algorithme plus coûteux en complexité mais pouvant prendre en compte des paramètres plus fins d'association. En effet le plus proche voisin que nous cherchons n'est pas forcément le plus proche voisin géométrique mais le plus proche voisin en terme de ressemblance.

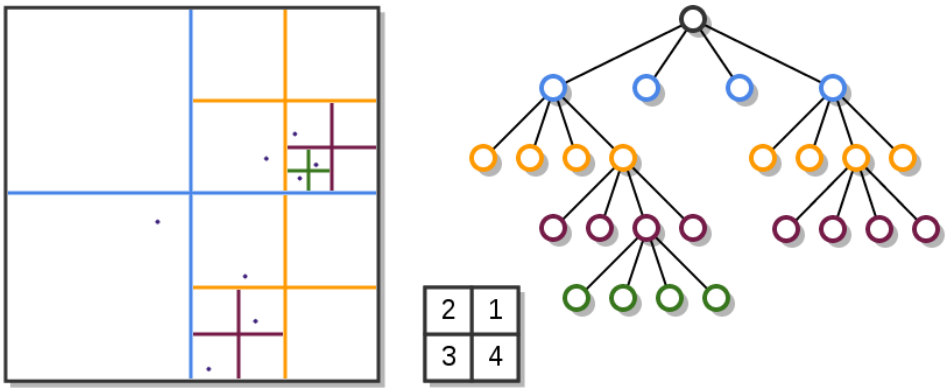


Figure 8.3 – Relation entre la géométrie de l'ensemble (le placement des points dans l'espace), et la structure de données générée, le quadtree. *Chaque découpage est effectué avec une couleur différente pour voir son association avec l'arbre. La convention d'ordre des fils est celle des quadrants d'un repère orthonormé. On observe le découpage des sous-espaces tant qu'il n'y a pas zéro ou un élément.*

Cette structure est généralisable facilement en 3 dimensions et portent le nom d'*oct-tree*, chaque nœud ne possède plus 4 mais 8 fils. Il est possible de passer en dimensions k en divisant en 2 chaque axe, et ainsi obtenir 2^k sous-espace, donc 2^k fils à chaque nœud.

Le problème de cette généralisation est la largeur et la profondeur de l'arbre obtenu, en effet si deux sources sont proches, il est parfois nécessaire d'effectuer un nombre important de divisions avant de réussir à séparer deux sources. C'est pour cette raison que cette algorithme ne sera utilisé que pour effectuer un premier partitionnement pour utiliser un algorithme plus lent sur des plus petites portions.

8.3.2 *kd-tree*

La division du *kd-tree* consiste à sub-diviser l'espace en tenant compte du nombre de points. En effet on remarque que le *quad-tree* est très rarement équilibré, de nombreux nœuds ne servent pas, etc. Ces problèmes ont une cause commune qui est la division selon des critères géométriques sans tenir compte des données que l'espace contient. Ainsi la division du *kd-tree* s'effectue suivant la médiane. Cette séparation permet d'obtenir deux sous-espaces contenant le même nombre de points plus ou moins 1 — si l'espace contient un nombre impair de points.

Le problème de la médiane est que cette séparation s'effectue dans une liste, donc un espace à 1 dimension. La solution consiste à choisir à chaque itération une dimension selon laquelle nous effecturons la division. Ce choix s'effectue selon la parité de l'itération,

si l'itération est pair nous effectuerons la division selon l'axe des x (ascension droite dans notre cas), si l'itération est impair selon l'axe des y (déclinaison).

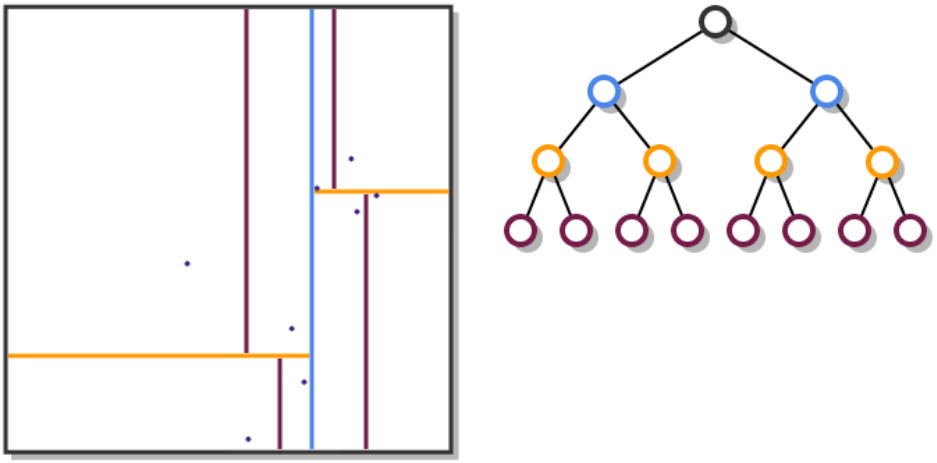


Figure 8.4 – Relation entre la géométrie de l'ensemble (le placement des points dans l'espace), et la structure de données générée, le kd-tree*. Chaque séparation divise l'espace en deux sous-espaces contenant un nombre égal de points. On remarque très facilement que l'arbre binaire est toujours équilibré grâce au découpage suivant la médiane.*

Pour le cas de k dimensions, le choix de la dimension s'effectue selon la congruence de l'itération modulo k, cela permet d'avoir une division régulière sur toutes les dimensions. La structure du *kd-tree* évolue très bien au passage des dimensions supérieures puisqu'il s'agit toujours d'un arbre binaire dont la profondeur n'est que fonction du nombre de données. Quelque soit le nombre de dimensions la profondeur reste égale à $\log_2(n)$ où n représente le nombre de points.

9 Dérécursification

L'algorithme 8.1 est un algorithme dit récursif, cela signifie qu'il fait appelle à lui-même dans sa résolution. C'est le cas par exemple de la définition de la fonction factorielle.

$$\begin{cases} n\,! = n \times (n-1)\,! \\ 0\,! = 1 \end{cases}$$

Lors de l'implémentation d'une telle fonction, celle-ci se rappelle elle-même jusqu'à un cas que l'on sait traiter c'est à dire factorielle de zéro. D'un point de vu informatique cela indique que nous ajoutons un appel de fonction dans la pile d'exécution. L'ajout

d'information dans la pile d'exécution nuit à la performance mais aussi à la modularité du programme. Pour ces raisons il peut être intéressant de récursifier un algorithme. Cette technique revient à gérer soit même la pile d'exécution en ne sauvant pas tout l'appelle à la fonction mais seulement son résultat intermédiaire. Le résultat final est obtenu en dépilant les informations préalablement stockée dans la pile.

9.1 Objectif

Notre algorithme est plus simple a comprendre sous sa forme récursive, et les seuls modifications que nous voulons apporter sont des limitations quant au nombre d'itérations. Ainsi l'aspect modulaire de la dérécursification ne nous intéresse pas, seul l'aspect performance nous intéresse.

Troisième partie

Résultats et discussion

10 Base de données

10.1 Le retour client

10.2 Efficacité

Étude de benchmark avec quelques requêtes régulièrement effectuée. Récupérer un csv avec les résultats et en sortir un joli boxplot avec R générer au moment de la compilation.

Voir fonction `BENCHMARK` de MySQL :

```
BENCHMARK(count,expr);
```

Le problème est que cette fonction ne permet d'être utilisé que sur des expression (`expr`) ne retournant qu'une ligne ou qu'une colonne.

10.3 Maintenabilité

La base de données est documenté (voir annexe) donc c'est cool.

11 Algorithmes de comparaison

11.1 Efficacité

11.1.1 Temps d'exécution

Utilisation de R pour la sortie (boxplot)

11.1.2 Utilisation mémoire

Utilisation de R pour la sortie (boxplot)

11.2 Crédibilité du programme

Comparaison des vrais positifs, faux négatifs et vrais négatifs, faux positifs.

12 Analyse des résultats

12.1 Résultat de la comparaison

12.2 Conclusion sur le *Stack*

13 Conclusion

This is the end

Bibliographie & Webographie