



# Rapport de TP

Module : Swarm Intelligence

Master 1 SII

TP

**Résolution du problème de satisfiabilité  
avec des méthodes constructives  
(Approche par espace des états)**

- Réalisé par :

**BENHADDAD Wissam**

**BOURAHLA Yasser**

23-02-2018

# Table des matières

<b>1</b>	<b>Introduction :</b>	<b>2</b>
1.1	Problématique :	2
1.2	Définitions	2
1.2.1	Problème SAT	2
1.2.2	Stratégie de recherche dans l'espace des états	3
1.2.3	Stratégie de recherche aveugle	3
1.2.4	Stratégie de recherche guidée	4
<b>2</b>	<b>Implémentation</b>	<b>6</b>
2.1	Structures de données	6
2.2	Conception et pseudo-code	6
2.2.1	Profondeur d'abord	6
2.2.2	Largeur d'abord	6
2.2.3	Recherche gloutonne	6
2.2.4	Algorithme A*	6
<b>3</b>	<b>Expérimentations</b>	<b>7</b>
3.1	Donnés	7
3.2	Environement de travail	8
3.2.1	Machines	8
3.2.2	Outils utilisés	9
3.3	Résultats	9
3.4	Statistiques	9
3.5	Comparaison entres les quatres méthodes	9
<b>4</b>	<b>Conclusion</b>	<b>10</b>

# Chapitre 1

## Introduction :

### 1.1 Problématique :

Dans ce TP, nous allons tenter d'implémenter et de comparer plusieurs méthodes aveugles, dites aussi à **base d'espace d'états**, Pour la résolution du problème de satisfiabilité, plus communément appelé **Problème SAT**, Ce travail est aussi une application directe des différentes méthodes vues durant le premier semestre en ce qui concerne la **Résolution de problèmes**, mais aussi la **Complexité des algorithmes et les structures de données**.

### 1.2 Définitions

#### 1.2.1 Problème SAT

Dans le domaine de l'informatique et de la logique, le problème de satisfiabilité (**SAT**), est un problème de décision ou il s'agit de donner des valeurs de vérité à des littéraux<sup>1</sup> tel qu'un ensemble de clauses en forme normale conjonctive FNC<sup>2</sup> préalablement défini soit satisfiable, en d'autres termes, que toutes les clauses soient vraies pour les mêmes valeurs de vérité de leurs littéraux, ce problème est le premier à avoir été démontré comme étant **NP-Complet**, et cela par **Stephen Cook** dans [1], et qui a donc posé les fondements de l'informatique théoriques et de la théorie de la complexité.

---

1. Une variable logique ou bien sa négation

2. Une conjonction de disjonction de littéraux

## 1.2.2 Stratégie de recherche dans l'espace des états

En considérant l'espace de recherche comme étant un arbre, dont les noeuds sont les différents états du problème, nous pouvons classer les différentes stratégies de recherches en deux grandes catégories :

## 1.2.3 Stratégie de recherche aveugle

### Par profondeur d'abord (DFS)

L'algorithme de parcours en profondeur d'abord consiste à visiter un noeud de départ (souvent appelé **racine**), puis ensuite visiter le premier sommet voisin (ou **successeur**) jusqu'à ce qu'une profondeur limite soit atteinte ou bien qu'il n'y ait plus de voisin à développer, une variante de cet algorithme utilise deux ensembles **Open** et **Closed** qui représentent respectivement l'ensemble des noeuds du graphe qui n'ont pas encore été développés et ceux déjà développés, cet ajout permet à l'algorithme d'éviter de boucler indéfiniment sur un ensemble de noeuds.

### En Largeur d'abord (BFS)

Cet algorithme diffère de son prédécesseur par le fait qu'il visite tous les voisins (**successeurs**) d'un noeud avant de passer au noeud suivant, ce qui revient à gérer l'ajout et la suppression de l'ensemble Open comme une file, donc en mode **FIFO** (En supposant bien sûr qu'on dispose des deux ensembles open et closed), cet approche permet de sauvegarder tous les noeuds précédemment visités durant la recherche, ce qui peut causer un débordement de la mémoire lors de l'exécution sur machine (Ce point sera rediscuté dans [3.3](#) )

### Par coût uniforme

Le principe est simple, au fur et à mesure que l'algorithme avance et développe des noeuds, il garde en mémoire le coût <sup>1</sup>, le noeud qui sera ensuite choisi sera celui dont le coût accumulé est le plus bas, assurant ainsi de toujours choisir le chemin le plus optimal, si le coût pour passer d'un noeud à n'importe quel autre de ses voisins est le même quelque soit le noeud, l'algorithme est alors équivalent à celui de la recherche en largeur d'abord

---

1. Fonction retournant le coût pour passer du noeud de départ (la racine) au noeud courant

## 1.2.4 Stratégie de recherche guidée

### Recherche gloutonne (Greedy algorithm)

Cet algorithme est basé sur la notion d'heuristique<sup>1</sup>, au lieu de parcourir de façon "naïve" l'ensemble des noeuds dans l'espace de recherche, il choisit à chaque itération sur l'ensemble **open** le noeud le plus **prometteur** en terme de distance par rapport au but recherché.

---

1. Une fonction d'estimation de la distance séparant le noeud courant au but

## Algorithme A\*

Contrairement aux précédents algorithmes de recherche qui effectuaient une recherche de façon "naïve", l'algorithme A\* propose une vision un peu nouvelle, il utilise la notion de coût et celle d'heuristique, la fonction d'évaluation  $f$  est donc définie comme étant la somme de deux fonctions  $g$  et  $h$  ou :

- $g$  est la fonction qui retourne le cout d'un noeud  $n$
- $h$  est la fonction qui estime le cout d'un noeud  $n$  vers le but

Le principe de l'algorithme est donc de prendre le noeud dans **open** qui possède la valeur minimal de  $f$ , assurant ainsi de trouver le chemin optimal **ssi**. l'heuristique  $h$  choisie est admissible<sup>2</sup>

---

2. Ne surestime jamais le coût réel pour atteindre le but, elle est **optimiste**

# Chapitre 2

## Implémentation

### 2.1 Structures de données

### 2.2 Conception et pseudo-code

#### 2.2.1 Profondeur d'abord

#### 2.2.2 Largeur d'abord

#### 2.2.3 Recherche gloutonne

#### 2.2.4 Algorithme A\*

# Chapitre 3

## Expérimentations

### 3.1 Données



## 3.2 Environnement de travail

### 3.2.1 Machines

Pour les tests nous avons utilisé deux machines pour chaque groupes d'instances, autrement dit une machine pour effectuer les tests sur un ensemble d'instances satisfiables [UF75-325\[2\]](#) et une autre sur les instances contradictoires (non satisfiables) [UUF75-325\[2\]](#), les caractéristiques de chaque machine sont données dans les figures [3.1](#) et [3.2](#) suivantes :



FIGURE 3.1 – Machine A pour les instances contradictoires

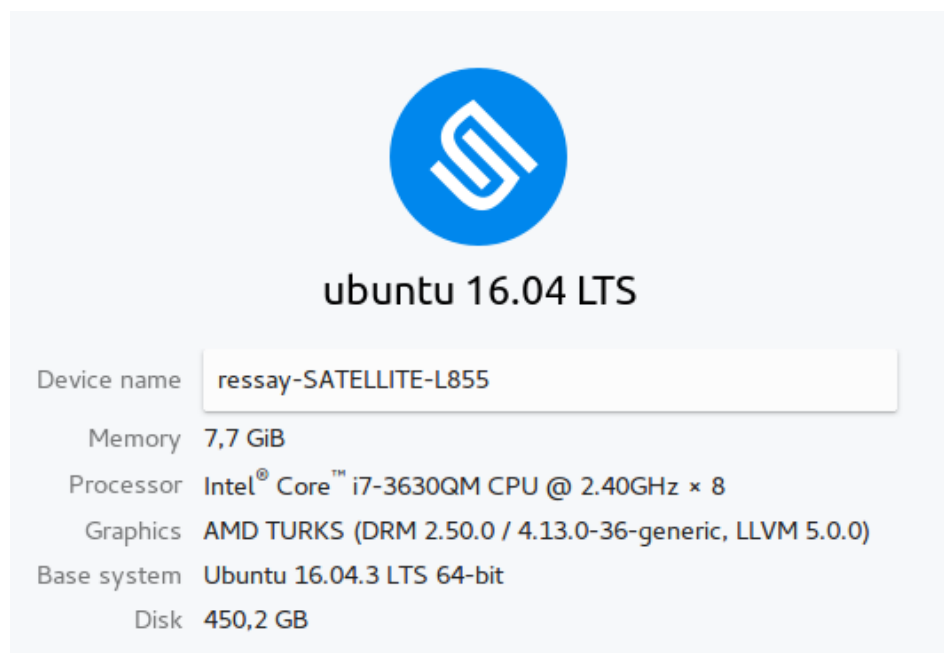


FIGURE 3.2 – Machine B pour les instances satisfiables

### 3.2.2 Outils utilisés

Langage de programmation :

IDE :

**IntelliJ Idea** L'environnement de développement choisit est [IntelliJ IDEA](#), spécialement dédié au développement en utilisant le langage [JAVA](#), il est proposé par l'entreprise [JetBrains](#) il est caractérisé par sa forte simplicité d'utilisation et les nombreuses plugins et extensions qui lui sont dédiées.

## 3.3 Résultats

## 3.4 Statistiques

## 3.5 Comparaison entre les quatre méthodes

## **Chapitre 4**

## **Conclusion**

# Table des figures

3.1	Machine <b>A</b> pour les instances contradictoires . . . . .	8
3.2	Machine <b>B</b> pour les instances satisfiables . . . . .	8

# Bibliographie

- [1] Stephen A. Cook. *The Complexity of Theorem-proving Procedures*. 1971.
- [2] **SATLIB** - benchmark for **SAT** problems. <http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html>.