Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'électronique et d'informatique
Département d'informatique

Rapport de TP

Module : Swarm Intelligence

Master 1 SII

TP

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

- Réalisé par :

  BENHADDAD Wissam
  BOURAHLA Yasser

23-02-2018

Table des matières

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 satisfiablité (**SAT**), est un problème de décision ou il s'agit d'assigner des valeurs de vérité à des literaux [1] tel qu'un ensembles de clauses en forme normale conjonctives FNC [2] préalablement défini soit satisfiable, en d'autres termes, que toutes les clauses soient vraies pour les mêmes valeurs de vérité de leurs literaux, 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'informathique théoriques et de la théorie de la complexité.

1. Une variables logique ou bien sa négation
2. Une conjonction de disjonction de litéraux

1.2.2 Stratégie de recherche dans l'espace des états

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

1.2.3 Stratégie de rercherche aveugle

Par prodonfeur d'abord (DFS)

L'algorithme de parcours en profondeur d'abord consiste a visité un noeud de départ (souvent appelé **racine**), puis ensuite visite 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 developper, une variante de cet algorithme utilise deux ensemble **Open** et **Closed** qui représentent réspectivement l'ensemble des noeuds du graphe qui n'ont pas encore étés developpés 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 prédecesseur 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 précèdemment visité 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 mesur que l'algorithme avance et développe des noeuds, il garde en mémoire le coût [1], 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 neoud, l'algorithme est alors équivalent à celui de la recherche en largeur d'abord

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

1.2.4 Stratégie de rercherche guidée

Recherche gloutonne (Greedy algorithm)

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

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

Algorithme A*

Contrairement aux précedents algorithmes de recherche qui effectuaient une recherche de façon "naîve", 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 [2]

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

Chapitre 2

Implémentation

2.1 Structures de données

La stratégie de recherche avec graphe requiert une représentation des entrées du problème, des états construisant une solution potentielle à ce dernier ainsi que le développement de ces états.

2.1.1 Représentation du problème SAT

Une instance du problème SAT peut être considérée comme un ensemble de clauses, chacune de ces clauses est une disjonction de littéraux. Dans ce rapport Nous proposons deux structures différentes pour les représenter que nous comparerons par la suite.

Représentation matricielle

Une première représentation serait d'associer à chaque clause de l'instance un tableau de taille égale au nombre de variables logiques utilisés dont la i$^{\text{ième}}$ case aura la valeur 1 si la variable *i* est présente dans la clause, -1 si sa négation est présente, 0 sinon. Ainsi en représentant toutes les clauses on obtient une matrice dont chaque ligne est associée à une clause.
L'exemple suivant montre une instance du problème SAT et sa représentation matricielle :

$$x_1 \vee \neg x_2 \vee x_5$$
$$\neg x_2 \vee x_4 \vee x_5$$
$$\neg x_1 \vee x_2 \vee \neg x_3$$

ces clauses vont être représentée comme suit :

1	-1	0	0	1
0	-1	0	1	1
-1	1	1	0	0

Représentation par *Bitset*

On pourrait aussi aborder la représentation du point de vu littéral, c'est à dire associer pour chaque littéral les clauses dans lesquels il est présent. Pour cela un tableau de bits appelé *Bitset* pourrait être utilisé où chaque bit *i* aurait la valeur 1 si la $i^{ième}$ clause contient le littéral, la valeur 0 sinon. On obtient donc un tableau de taille 2 fois le nombre de variables utilisés dont les entrés représentent les *Bitsets* des littéraux.
Pour le même exemple vu précédemment on obtient les *Bitsets* suivants :

x_1	1	0	0
x_2	0	0	0
x_3	0	0	0
x_4	0	1	0
x_5	1	1	0

$\neg x_1$	0	0	1
$\neg x_2$	1	1	0
$\neg x_3$	0	0	1
$\neg x_4$	0	0	0
$\neg x_5$	0	0	0

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 Donnés

3.2 Environement de travail

3.2.1 Machines

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

FIGURE 3.1 – Machine **A** pour les instances contradictoires

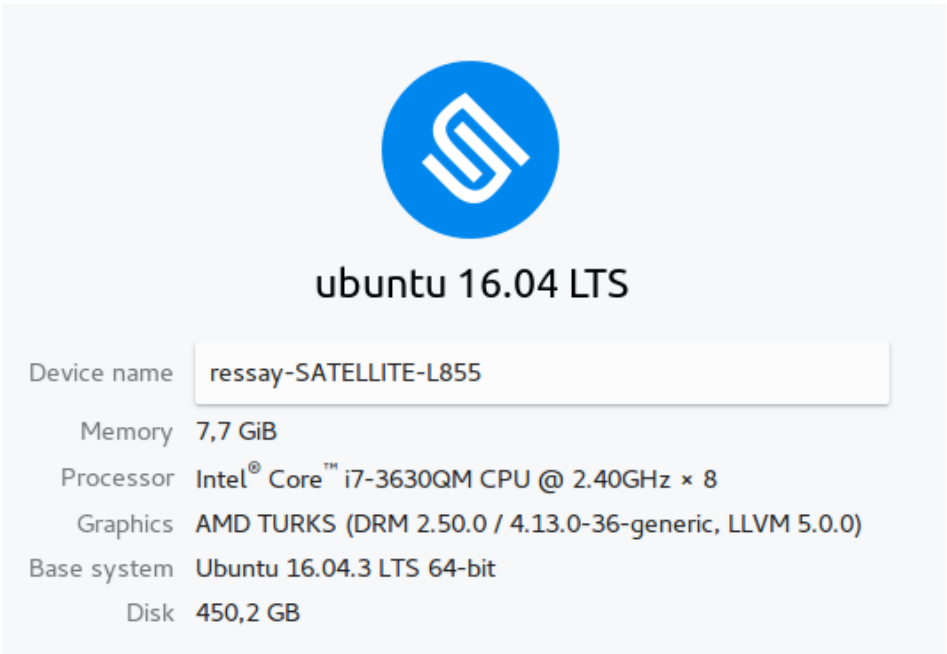

FIGURE 3.2 – Machine **B** pour les instances satisfiables

3.2.2 Outils utilisés

Langage de programmation :

IDE :

IntelliJ Idea L'environement de dévelopement 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 nombreuse plugins et extentions qui lui sont dédiées.

3.3 Résultats

3.4 Statistiques

3.5 Comparaison entres les quatres méthodes

Chapitre 4

Conclusion

Table des figures

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.