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 Informatique

Master 1 SII

Projet Intelligence en essaim

Rapport de la partie 1

Résolution du problème SAT avec l'approche espace des états

Réalisé par :

- AYAD Abdou 171731057441
- GUENDOUZ Akrem Mohamed 171731070870
- BOUSSEKSOU Yacine 171731078048

30 mai 2021

Sommaire

1 Introduction

Le problème SAT est un problème de satisfaisabilité booléenne fondé sur la logique propositionnelle, encore appelée logique d'ordre 0 ou logique des prédicats.

Le problème de satisfiabilité SAT est le premier problème qui a été montré NP-complet par Stephan Cook et Leonid Levin (théorème de Cook en 1971). L'étude des variantes de ce problème permet d'une part d'introduire des techniques de preuves et d'autre part de pointer sur la frontière entre problèmes faciles et difficiles.

En effet, le problème SAT est représenté par une instance caractérisée par un nombre de variables qui appartiennent à un ensemble de clauses, chacune composée de littéraux, sous forme normale conjonctive (FNC). Le but (la solution) est de trouver une distribution (assignation de valeurs booléennes) de variables rendant toutes les clauses vraies dans la version originale du problème ou à maximiser les clauses satisfaites dans le cas du Max-SAT.

Dans cette première partie, nous nous intéressons à deux stratégies : La recherche aveugle et la recherche informée, utilisant des heuristiques. Il s'agit d'implémenter des algorithmes de résolution du problème de satisfiabilité basés sur les deux méthodes citées précédemment, puis les tester sur des fichiers de benchmarks satisfiables et d'autres non- satisfiables, afin de comparer leurs complexité et efficacité.

2 Approche Espace des états

2.1 Méthodes aveugles

Les techniques de recherche aveugle sont basées sur la vérification de toutes les solutions possibles d'un problème. Ainsi, toutes les combinaisons sont examinées aveuglement, sans aucune information aidante à la prise de décision. Ces approches sont les plus basiques pour résoudre les problèmes dans l'intelligence artificielle.

2.1.1 Recherche en largeur d'abord (BFS)

2.1.1.1 Présentation

Le parcours en largeur (en anglais : BFS ou Breadth First Search) est l'un des types de recherche aveugle qui consiste a parcourir l'arbre niveau par niveau ; explorer tous les nœuds de l'arbre en allant du nœud source (racine) vers le dernier niveau (les feuilles) et de gauche vers la droite pour chaque niveau.

Malgré l'énorme consommation de l'espace mémoire, cette méthode garantira de trouver des solutions si elles existent.

2.1.1.2 Structures et implémentation

Le parcours en largeur est basé sur le principe FIFO (First In First Out) est pour réaliser cela on utilise une file.

Cette file est une liste chaînée nommée Open, sert à stocker les nœuds crées tout au long de la recherche de telle sort l'insertion des nouveaux nœuds se fait à la fin (queue) et la suppression se fait au début (tête).

Le principe de fonctionnement de cette file est qu'avant chaque suppression de la tête (défilement), l'insertion (enfilement) de tous ses nœuds adjacents s'effectue pour garder toujours l'ordre des nœuds pour ce parcours.

Le calcul de la complexité empirique temporelle et spatiale:

	complexité empirique temporelle	complexité empirique spatiale
Complexité au meilleur cas	le but est atteint en une seule itération. La complexité en notation asymptotique de Landau est constante, égale à : $C(n) = O(1)$.	Elle est évaluée en fonction de la taille (nombre d'éléments) de la liste "open", comme suit : Le but est atteint dés la première itération, aucun élément (nœud) n'est mis dans "open". Du coup, la complexité est constante, égale à : $C(n) = O(1)$.
Complexité au pire des cas	toutes les possibilités seront parcourues par la boucle. La complexité en notation asymptotique de Landau est exponentielle, égale à : $C(n) = O(2^n)$ ("n" étant le nombre de variables du problème 3-SAT).	toutes les combinaisons seront essayées. De ce fait, la complexité est exponentielle, égale à : $C(n) = O(2^n)$.

2.1.2 Recherche en profondeur d'abord (DFS)

2.1.2.1 Présentation

La recherche en profondeur d'abord (DFS) est un algorithme, aussi classé parmi les méthodes aveugles, permettant de parcourir ou de rechercher des structures de données arborescentes. L'algorithme commence au nœud racine (en sélectionnant un nœud arbitraire comme nœud racine dans le cas d'un graphe) et explore autant que possible le long de chaque branche avant de revenir en arrière.

Notons que cette méthode a une complexité spéciale meilleure qu'en BFS car elle se concerne d'une branche de l'arbre à la fois.

2.1.2.2 Structures et implémentation

La méthode DFS, contrairement au BFS, utilise la structure "pile" pour le sauvegarde des nœuds. De cette manière, le principe LIFO est conservé. De plus, on utilise une liste "open" pour les nœuds à explorer.

Pour le problème 3-SAT, l'implémentation de "open" est un petit peu différente qu'au BFS. Cette fois ci, un nœud représente l'ensemble de 3 informations:

- Valeur : Le littéral (numéro de variable, positif ou négatif).
- Antécédent : La valeur du père.
- Numéro du fils : Spécifie si c'est le premier fils de l'antécédent ou bien le deuxième.

Ce choix économise l'espace mémoire utilisé dans la recherche car trois entiers sont retenus par nœud au lieu d'une solution complète.

Le calcul de la complexité empirique temporelle et spatiale:

	complexité empirique temporelle	complexité empirique spatiale
Complexité au meilleur cas	le but est atteint en une seule itération. La complexité en notation asymptotique de Landau est constante, égale à : $C(n) = O(1)$.	Elle est évaluée en fonction de la taille (nombre d'éléments) de la liste "open", comme suit : Le but est atteint dés la première itération, aucun élément (nœud) n'est mis dans "open". Du coup, la complexité est constante, égale à : $C(n) = O(1)$.
Complexité au pire des cas	toutes les possibilités seront parcourues par la boucle. La complexité en notation asymptotique de Landau est exponentielle, égale à : $C(n) = O(2^n)$ ("n" étant le nombre de variables du problème 3-SAT).	t Toutes les combinaisons sont essayées. Chaque itération rajoute deux nouveaux nœuds (un de la branche courante, l'autre à traiter ultérieurement) à "open". Ainsi, la taille maximale de "open" est atteinte au dernier niveau (numéro "n") où la branche actuelle est explorée complètement. Alors, la complexité est linéaire, égale à : $2n \rightarrow C(n) = O(n)$.

2.2 Méthodes heuristiques

Les méthodes de recherche heuristiques représentent une amélioration des approches aveugles vues en-dessus. Ces dernières, malgré qu'elles sont simples à implémenter, mais elles ont des inconvénients, parmi ces inconvénients on cite le problème d'explosion combinatoire qui nécessite beaucoup de temps pour la mise en marche de ces heuristiques. Par conséquent, les heuristiques présentent un avantage grâce aux fonctions utilisées et qui mènent à des résultats plus rapides. En revanche, pour choisir l'heuristique adéquate il faut bien comprendre le problème posé afin d'aboutir à des résultats efficaces.

2.2.1 Algorithme A*

2.2.1.1 Présentation

Les algorithmes A-étoile(A-Star), utilisés très souvent dans la résolution des problèmes ou dans le domaine de l'IA, combinent l'efficacité de la recherche avare et l'optimalité de la recherche à coût uniforme. Ils sont définis par une fonction d'évaluation f(n) = g(n) +h(n) où :

- la variable "n" désigne le nœud courant.
- g(n) : Le coût du chemin allant de l'état initial jusqu'à le nœud "n".
- h(n) : Une estimation du coût du chemin reliant le nœud "n" à un état final.

2.2.1.2 Structures et implémentation

Comme les méthodes aveugles, le A* est une recherche arborescente qui utilise "open" pour stocker les nouveaux nœuds créés, la différence est que cette liste est triée en ordre croissant selon la fonction "f". Ainsi, le prochaine exploration dépend de l'évaluation heuristique qui consiste à estimer le meilleur chemin pour atteindre le but.

Pour le problème 3-SAT, un nœud représente une solution complète. En ce qui concerne le choix des mesures des heuristiques, on les a défini comme suit :

- g(n) : Le nombre de clauses satisfaites par le nœud courant.

- h(n) : Le nombre de clauses non-satisfaites par le nœud courant.

Le calcul de la complexité empirique temporelle et spatiale:

	complexité empirique temporelle	complexité empirique spatiale
Complexité au meilleur cas	le but est atteint en une seule itération. La complexité en notation asymptotique de Landau est constante, égale à : $C(n) = O(1)$.	Elle est évaluée en fonction de la taille (nombre d'éléments) de la liste "open", comme suit : Le but est atteint dés la première itération, aucun élément (nœud) n'est mis dans "open". Du coup, la complexité est constante, égale à : $C(n) = O(1)$.
Complexité au pire des cas	toutes les possibilités seront parcourues par la boucle. La complexité en notation asymptotique de Landau est exponentielle, égale à : $C(n) = O(2^n)$ ("n" étant le nombre de variables du problème 3-SAT).	toutes les combinaisons seront essayées. De ce fait, la complexité est exponentielle, égale à : $C(n) = O(2^n)$.

3 Le problème SAT

3.1 Description

Le SAT est un problème de décision et de satisfiabilité booléenne qui est basé sur la logique propositionnel et sur son lexique. Pour cela, on présente les définitions de la terminologie utilisée dans ce projet :

- Instance SAT : une conjonction d'un ensemble fini de clauses.

- Variable propositionnelle : Une variable booléenne qui peut être soit vraie ou fausse, elle est l'élément fondamental des formules propositionnelles.

- Formule propositionnelle : Une expression construite à partir de connecteurs et de variables propositionnelles.

- Littéral : Une variable propositionnelle aussi appelé atome (littéral positif) ou sa négation (littéral négatif).

- Clause : Une disjonction de littéraux (forme normale disjonctive). Ainsi, le problème SAT est une conjonction d'un ensemble fini de clauses.

- Une instance 3-SAT : une conjonction d'un ensemble de clauses formé de 3 littéraux exactement.

3.2 Structure de données

Le choix de la structure de donnée pour représenter ce problème SAT a un grand impact sur les résultats (en terme de performance) et l'un des facteurs majeurs pour aller le plus loin possible dans la résolution de ce problème.

Pour bien modéliser ce problème on a opté pour une matrice dynamique (ArrayList of ArrayList) .

-pour modéliser une clause, on a conçu un vecteur dynamique qui contient dans chaque case un littéral (la valeur de l'indice du littéral ; ça peut être positive ou négative).

$C_1 = X_5 \vee \overline{X_3} \vee X_1$ ==>

5	-3	1

-pour modéliser l'ensemble des clauses, on a implémenté aussi un vecteur dynamique qui contient dans chaque case une clause.

$C_1 = X_5 \vee \overline{X_3} \vee X_1$

$C_2 = X_4 \vee \overline{X_2} \vee \overline{X_9}$

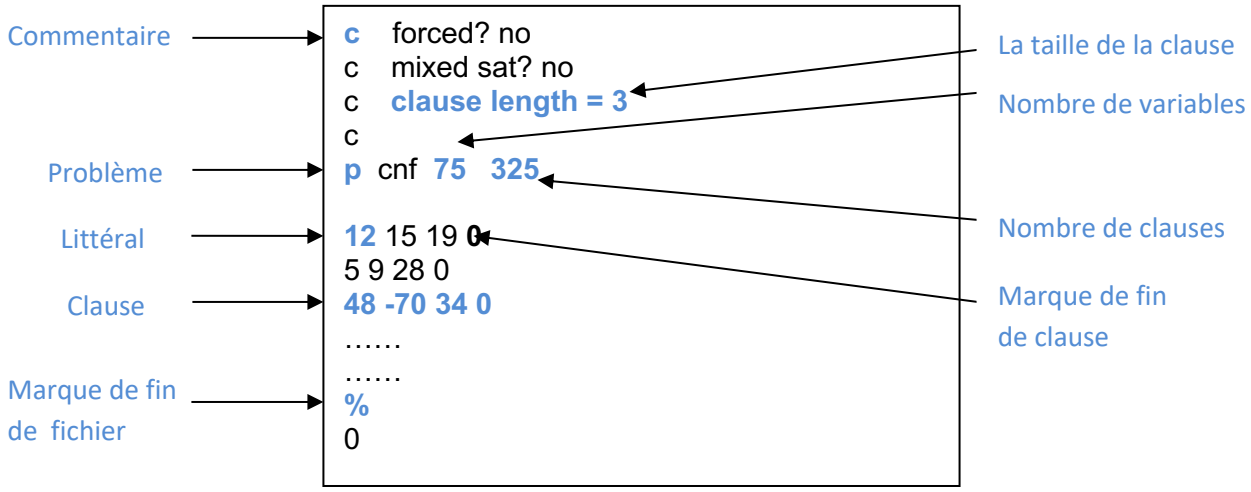
$C_3 = \overline{X_{41}} \vee \overline{X_{62}} \vee \overline{X_1}$

==>

5	-3	1
4	-2	-9
-41	-62	-1

3.3 Description d'un fichier CNF

Une instance du problème 3-SAT est contenue dans un fichier de type DIMACS CNF. Ce dernier est utilisé pour définir une expression booléenne écrite en forme normale conjonctive, en format ASCII, se présente comme suit :



Le fichier CNF est constitué par les commentaires qui se sont caractérisés par le « c » au début de la ligne est qui contiens plusieurs informations sur le fichier, la ligne problème qui commence par le « p » est qui donne le nombre de clauses et le nombre de variables et enfin on a les clauses (disjonction des littéraux).

4 Expérimentations

4.1 Description du Benchmark

Les données utilisées pour tester les différents algorithmes implémentés pour SAT sont accessibles à partir du lien suivant : https ://www.cs.ubc.ca/ hoos/SATLIB/benchm.html.

Les fichiers utilisés pour les tests sont les benchmarks uf75-325 et uuf75-325. uf75-325 sont des instances satisfiables alors que uuf75-325 sont des instances contradictoires. Ces benchmarks comportent chacun :

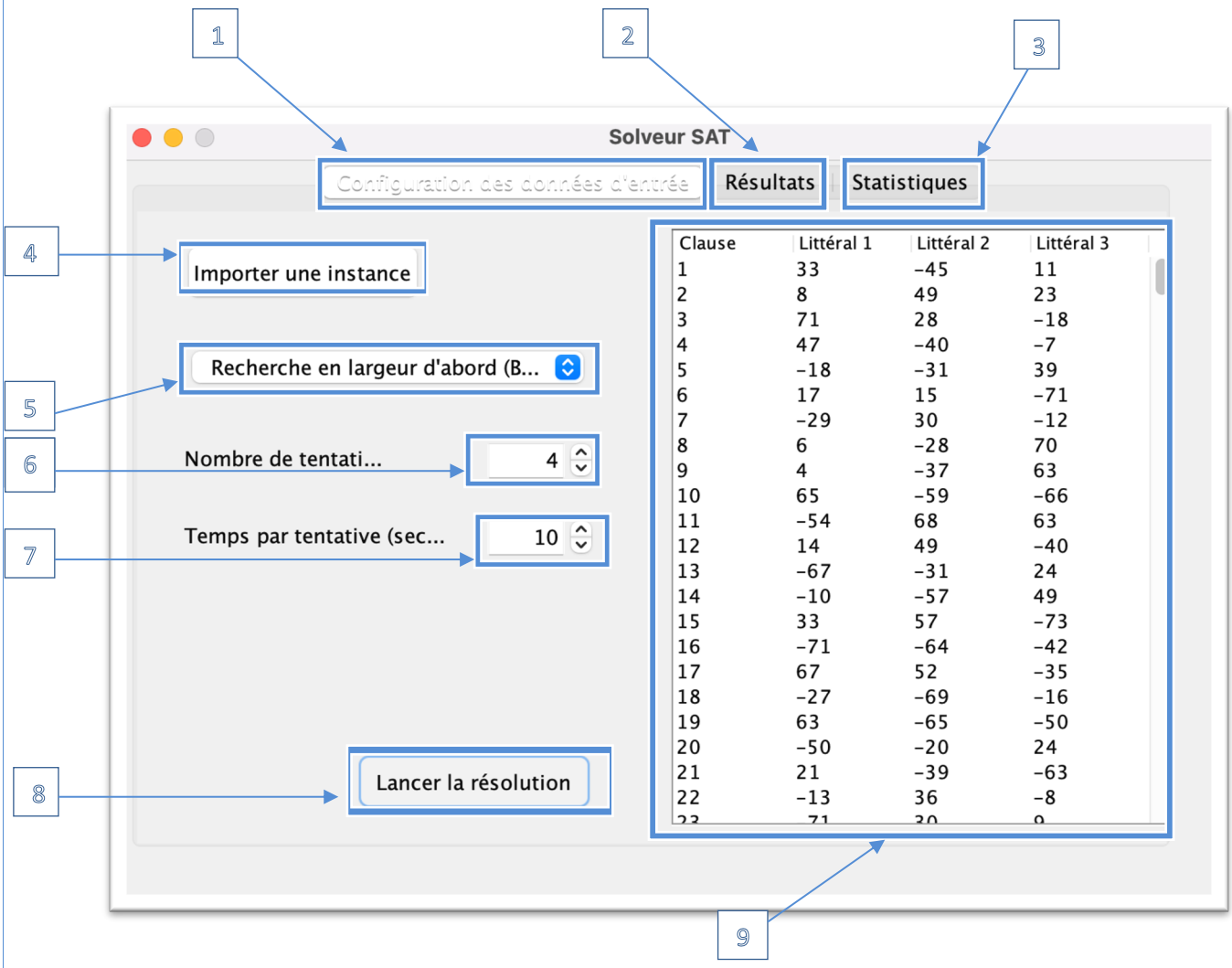
- 100 instances de type 3-SAT en format CNF.
- Chaque clause est composée de 3 variables (taille de clause = 3).
- Chaque instance contient 325 clauses et 75 variables.

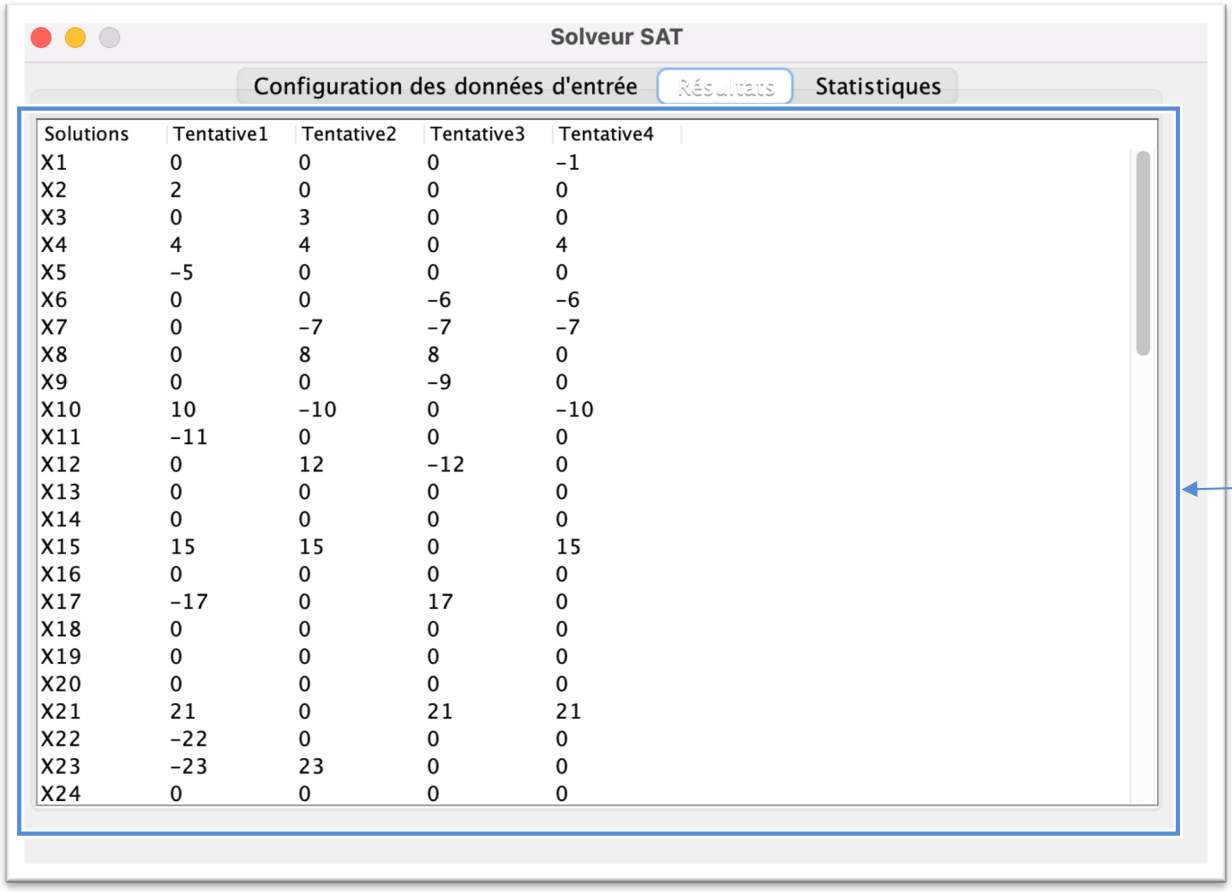
4.2 Environnement expérimental

- RAM : 8GO1867 MHz DDR3.
- Processeur : 2,7 GHz Intel Core i5 double cœur
- Carte Graphique : Intel Iris Graphics 6100 1536 Mo.
- Système d'exploitation : MacOS Big Sur.
- Langage de programmation : Java.
- Environnement de développement : IntelliJ.

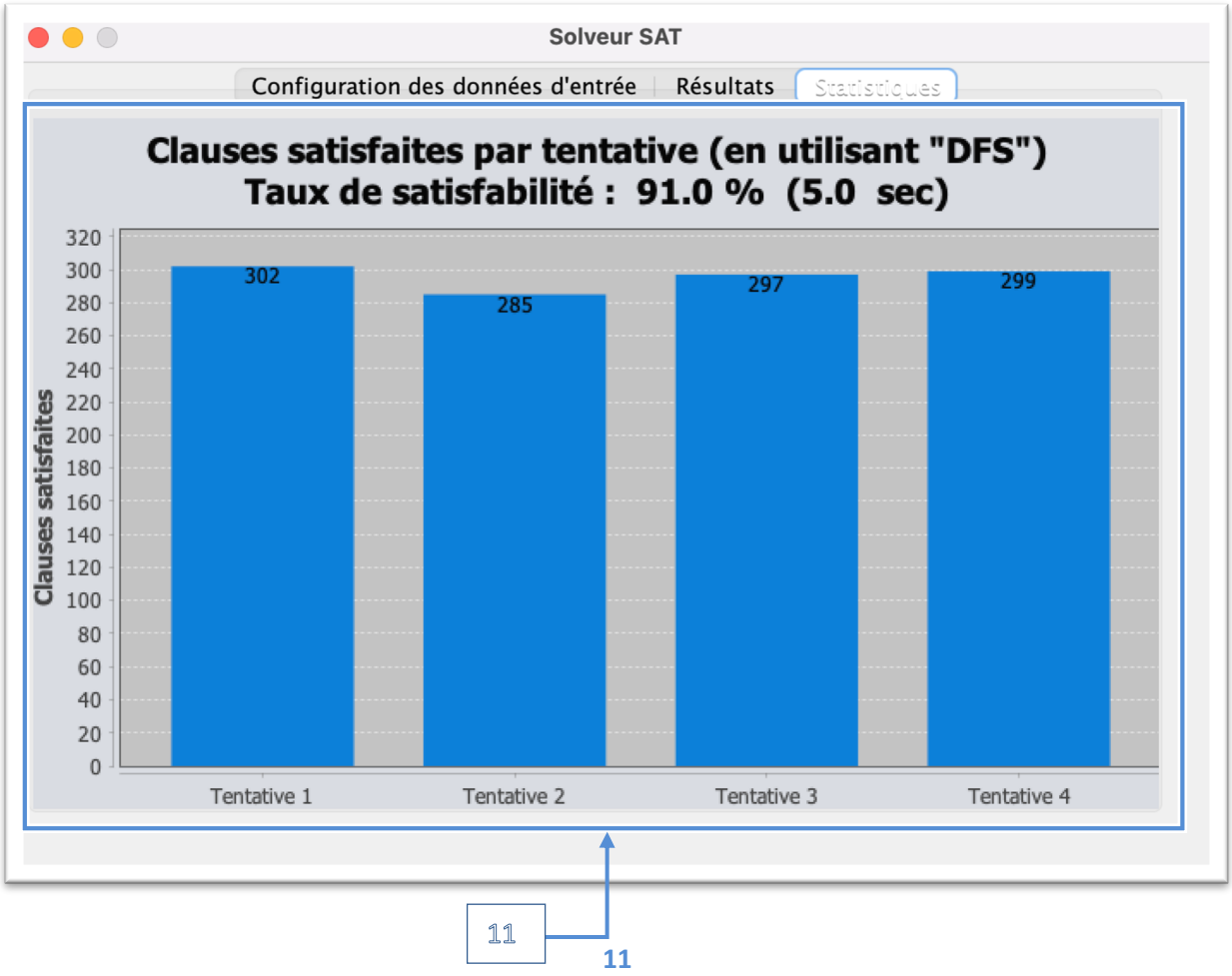
4.3 Interface graphique :

4.3.1 Design





Solutions	Tentative1	Tentative2	Tentative3	Tentative4
X1	0	0	0	-1
X2	2	0	0	0
X3	0	3	0	0
X4	4	4	0	4
X5	-5	0	0	0
X6	0	0	-6	-6
X7	0	-7	-7	-7
X8	0	8	8	0
X9	0	0	-9	0
X10	10	-10	0	-10
X11	-11	0	0	0
X12	0	12	-12	0
X13	0	0	0	0
X14	0	0	0	0
X15	15	15	0	15
X16	0	0	0	0
X17	-17	0	17	0
X18	0	0	0	0
X19	0	0	0	0
X20	0	0	0	0
X21	21	0	21	21
X22	-22	0	0	0
X23	-23	23	0	0
X24	0	0	0	0



4.3.2 Définition des fonctionnalités :

1 : Interface de configuration des données et paramètres d'entrée.
2 : Interface d'affichage des résultats.
3 : Interface d'affichage des statistiques des différentes approches.
4 : Bouton d'importation d'une instance du problème 3-SAT (fichier.cnf).
5 : Liste déroulante pour choisir une des méthodes à tester (BFS, DFS, A*).
6 : Spécification de nombre de tentatives.
7 : Spécification du temps d'exécution par tentative.
8 : Lancement de la recherche des solutions possibles.
9 : Affichage du fichier CNF à tester.
10 : Affichage des solutions possibles trouvées.
11 : Affichage des statistiques liées à la résolution du problème 3-SAT.

4.3.3 Résultats expérimentaux

Les deux fichiers "uf75-325" et "uuf75-325" sont maintenus avec 5 instances de test, et des paramètres (*nombre d'essais = 4*, *temps par essai = {5 ;10}secondes*). Le tableau suivant illustre les résultats obtenus pour chaque méthode :

Les testes de la recherche en profondeur d'abord

Numéro de l'instance	Nombre d'essais	Temps par essai (sec)	Taux moyen de satisfiabilité	Temps moyen d'exécution
uf75-01.cnf	4	5	92,1538	5
uuf75-010.cnf	4	10	91,92308	10
uf75-020.cnf	4	5	92,2307	5
uuf75-030.cnf	4	10	90,3076	10
uf75-060.cnf	4	5	92,923	5
La moyenne	4	7	91,907636	7

Les testes de la recherche en largeur d'abord

Numéro de l'instance	Nombre d'essais	Temps par essai (sec)	Taux moyen de satisfiabilité	Temps moyen d'exécution
uf75-01.cnf	4	5	41,0769	5
uuf75-010.cnf	4	10	43,7692	10
uf75-020.cnf	4	5	40,8461	5
uuf75-030.cnf	4	10	43,2307	10
uf75-060.cnf	4	5	41,6153	5
La moyenne	4	7	42,10764	7

Les testes de la méthode heuristique (A*)

Numéro de l'instance	Nombre d'essais	Temps par essai	Taux moyen de satisfiabilité	Temps moyen d'exécution
uf75-01.cnf	4	5	92,1538	5
uuf75-010.cnf	4	10	91,3846	10
uf75-020.cnf	4	5	92,8461	5
uuf75-030.cnf	4	10	92,6923	10
uf75-060.cnf	4	5	91,5384	5
La moyenne	4	7	92,1692	7

5 Conclusion

Apres avoir testé les différentes méthodes conçues et comparer leurs résultats, on déduit que la méthode heuristique (A*) est la méthode la plus performante par rapport aux méthodes aveugle (BFS – DFS).

La recherche en largeur est la méthode la moins efficace pour le problème de satisfiabilité du à sa complexité spatiale et pour la recherche en profondeur, on a constaté qu'elle est beaucoup plus efficace que la recherche en largeur en termes d'espace mémoire. Mais, elle reste beaucoup moins optimale que la recherche utilisant une heuristique en termes de temps.

En conclusion, la taille de l'espace des états, la structure de donnée utilisée pour la représentation et la méthode de recherche choisie se sont les facteurs qui ont une grande influence sur les performances et l'optimisation des résultats.