

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Département Informatique

Master 1 SII

Métaheuristique et algorithmes évolutionnaires

RAPPORT DU PROJET

**-Etude des algorithmes de recherches
aveugles et évolutionnaires pour résoudre un
problème SAT-**

Réalisé par :

Groupe 12 :

BOUCENNA	Abderaouf	171731055577	G1
NOUAR	Sofia	171731058542	G3
RAHAL	Arslane	171731044215	G3
KARA	AbdelKader Mehdi	161631080394	G3

2020/2021

TABLE DES MATIÈRES

INTRODUCTION	3
Partie I	4
-Utilisation des méthode aveugles et heuristiques pour résoudre un problème SAT-	4
1.1. Les méthodes de recherche	5
1.1.1. aveugle	5
Recherche en largeur d'abord BFS	5
Recherche en profondeur d'abord DFS	5
1.1.2. heuristiques	6
L'algorithme A*:	6
1.2. Étude des algorithmes de recherche aveugle et heuristiques pour résoudre un problème SAT	7
Langage de programmation : JAVA	7
Environnement de développement : Eclipse.	7
1.2.1. Pseudo algorithme des algorithmes de résolution utilisées :	7
L'algorithme de recherche en largeur d'abord :	7
Environnement de travail :	7
L'algorithme de recherche en profondeur d'abord :	8
L'algorithme A* :	9
1.2.2.Complexité spatiale et temporelle des algorithmes de résolution	10
1.3.Test des algorithmes de résolution	10
a- Description des données de test:	10
b- Les résultats obtenues :	11
Recherche en largeur d'abord :	11
Tableau englobant les résultats :	11
Graphe SAT :	12
Graphe UNSAT:	13
Recherche en profondeur d'abord :	13
Tableau englobant les résultats SAT:	13
Graphe SAT :	14
Tableau englobant les résultats UNSAT:	14
Graphe :	15
Recherche A* :	15
Tableau englobant les résultats UNSAT :	15
Graphe NON-SAT :	16

Tableau englobant les résultats SAT :	16
Graphe SAT:	17
1.4. Conclusion	17
1.5. INTERFACE GRAPHIQUE	17
Partie II	23
-Utilisation des algorithmique évolutionnaires pour résoudre le problème SAT-	23
Notions de bases	24
2.1.1. L'intelligence en essaim (Swarm intelligence)	24
2.1.2. Méta heuristique, swarm intelligence et algorithmes évolutionnaires	24
2.1. GA : L'algorithme génétique	25
2.1.1. Principe :	25
Sélection naturelle : est un principe qui permet d'optimiser les populations d'individus.	25
Croisement : reproduction des espèces sélectionnées entre eux.	25
Mutation : mutation des espèces issues du croisement.	25
2.1.2. Avantages :	26
2.1.3. Pseudo-algorithme pour résoudre MAX-SAT:	26
2.1.4. Mise à jour de l'interface:	28
2.1.5. Tests et résultats :	Error! Bookmark not defined.
tableau regroupant les différents paramètres de test	29
Tableau englobant les résultats des fichiers SAT	30
Discussion et conclusion	31
2.2. PSO : Algorithme d'optimisation de l'essaim de particules	32
2.2.1. Principe :	32
2.2.2. Avantages :	32
2.2.3. Pseudo-algorithme:	33
2.1.4. Mise à jour de l'interface:	34
2.2.5. Tests et résultats :	34
tableau regroupant les différents paramètres de test	35
Tableau englobant les résultats des fichiers SAT	35
Discussion et conclusion	36

INTRODUCTION

En informatique, le défi à relever se résume à résoudre un problème donné en un temps raisonnable tout en consommant un espace mémoire minimal.

Plusieurs méthodes ont été créées: les méthodes de **recherche aveugles** dites *non informées* qui englobent **les algorithmes de recherche en largeur et en profondeur d'abord**, qui explorent, au pire des cas, tout l'espace de recherche pour essayer de trouver une solution, et la méthode **aveugle informée** (algorithme A*) utilisant des heuristiques pour mieux orienter la recherche et essayer de retourner une solution plus ou moins optimale, en minimisant l'espace de recherche, quand l'heuristique est performante.

Néanmoins, ces algorithmes exigent beaucoup de temps et consomment beaucoup d'espace mémoire sans même être sûr de résoudre le problème à la fin.

Pour cela de nouvelles méthodes de recherche plus optimales ont été découvertes s'inspirant de l'intelligence du groupe et de la génétique humaine.

Ces algorithmes évolutionnaires ont permis d'optimiser en temps et en espace la résolution de problèmes complexes.

Le problème SAT est un problème NP-COMPLET composé d'un ensemble de formules conjonctives, appelées aussi clauses, qui essaye d'affecter des valeurs booléennes aux variables les contenant, qui rendent toutes les clauses vraies. pour que toutes les clauses de la formule soient vraies.

Le problème de satisfiabilité est aujourd'hui utilisé dans de nombreux domaines comme la cryptanalyse, la vérification de matériels et de logiciels...

Le but de cette première partie de ce projet est d'évaluer la performance de ces méthodes de recherches à résoudre un problème sat afin de réaliser leurs inefficacité quand le problème devient complexe. Les tests sont effectués sur des fichiers benchmarks satisfiables ou non satisfiables, nous notons à chaque fois le temps d'exécution.

Partie I

**-Utilisation des méthode aveugles
et heuristiques pour résoudre un
problème SAT-**

1.1. Les méthodes de recherche ¹

1.1.1. aveugle

Les méthodes de recherche aveugle sont des méthodes visant à découvrir le chemin qui mène à un nœud but en explorant au maximum l'arbre de recherche. Elles arrivent à trouver un tel chemin s'il existe. On distingue deux principaux algorithmes :

- Recherche en largeur d'abord BFS

Un parcours en largeur explore l'arbre à partir d'un sommet donné (racine), et crée ces fils de gauche à droite, sauvegardés dans une liste **Ouverte**, niveau par niveau, et la même stratégie est appliquée à ces nœuds fils générés. Chaque nœud traité est supprimé de cette liste.

Les nœuds de *Ouvert* (liste des nœuds feuilles de l'arbre de recherche qui n'ont pas encore été sélectionnés pour être développés) sont dans l'ordre croissant de leur profondeur.

Exemple (sur wikipédia): Pour $n = 10$, puissance de calcul 10 000 noeuds par seconde, espace de 1000 octets de mémoire pour un noeud :

Pour une profondeur 8, il faut 10^9 nœuds, 31 h de calcul et 1 Téra d'espace.

Pour une profondeur 12, il faut 10^{13} noeuds, 35 ans de calcul 10 Pétaoctets et 1024 Téra

La complexité en temps et espace est en $O(a^n)$ avec :

- a : le nombre maximum de successeurs d'un nœud.
- n : la profondeur du premier nœud qui résout le problème.

- Recherche en profondeur d'abord DFS

Le parcours en profondeur explore un chemin à la fois avec un retour arrière si la profondeur max est atteinte et la solution n'est pas trouver.

¹ référence : chapitre 4 stratégies de recherche, résolution de problème M1 SII, H. Azzoune.

On garde en mémoire (dans l'ensemble **Ouvert**) tous les nœuds (pour faire le retour arrière). Dans cet ensemble, les nœuds sont en ordre décroissant suivant leur profondeur dans l'arbre. Les plus profonds sont placés en 1ier.

La complexité en temps est en $O(a^n)$ et $O(a * n)$ en espace avec :

- a : le nombre maximum de successeurs d'un nœud.
- n : la profondeur maximale.

1.1.2. heuristiques

Les méthodes de recherche aveugle sont des méthodes exhaustives, malgré qu'elles arrivent à trouver un tel chemin s'il existe, elles sont coûteuses en mémoire car elles développent trop de nœuds avant d'arriver au but.

Comme alternative plus efficace que la recherche aveugle, on peut utiliser certaines informations appelées « informations heuristiques ».

Une heuristique est une connaissance qui sert à guider lors du choix d'un nœud.

Les procédures de recherche utilisant de telles informations sont dites «heuristiques ». Certaines heuristiques réduisent considérablement l'effort de recherche mais ne garantissent pas de trouver le chemin le moins coûteux.

- L'algorithme A*:

L'algorithme de recherche A* minimise le coût du nœud n , en attribuant des priorités selon une fonction (choisie par le programmeur) **dite d'évaluation**, qui permet de rendre un nœud plus prioritaire.

1.2. Étude des algorithmes de recherche aveugle et heuristiques pour résoudre un problème SAT

Langage de programmation : JAVA

Environnement de développement : Eclipse.

1.2.1. Pseudo algorithme des algorithmes de résolution utilisées :

L'algorithme de recherche en largeur d'abord :

Environnement de travail :

- . RAM : 8GO
- . Processeur : Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
- . Système d'exploitation : Windows 10-64 bits

```
Proc searchBFS() {
    input : open : contains root of the tree
           clauses : clauses to solve
    output : solution if found (variable instantiated), else nothing
    Var:
        int[] solution=null;
    Start{
        while (!open.isEmpty ){
            root.createLeftSon();
            LeftSon.value = 1;
            if (all clauses are SAT in this node){
                solution[LeftSon]= LeftSon.value;
                closeQ.add(solution)
                return solution;
            }
            else {
                open.add(LeftSon);
                root.createRightSon();
                RightSon.value = -1;
                if (all clauses are SAT in this node){
                    solution[RightSon]= RightSon.value;
                    open.remove(root);
                    closeQ.add(solution)
                    return solution;
                }
            }
        }
    }
```



```

    }
}
}
if(solution==null){
    print("no solution found");
}
}
}

```

L'algorithme de recherche en profondeur d'abord :

Environnement de travail :

RAM : 16 Gb

Processeur : Intel Core i7-6700k

Système d'exploitation : Windows 10-64 bits

```

public void DFS() {
    input : Node ROOT,
           Vector<Clause> clVect
    output : Vector<SAT> Open : contain all solution found,
           int[] maxSat

    Var:
    int inst;

    //Looking for the value not instensiat
    inst = ROOT.getContent().Randinst();
    //Verify if the value is legit
    if(inst>0) {
        CreateLeftSon();
        if (ALL clause are sat)
        {
            Ouvert.add(solution);
        }
        if (ROOT.getLS().Numsat>maxSat) {
            maxSat= ROOT.Numsat;
        }
        ROOT.getLS().search();
        CreateRightSon();
        if (ALL clause are sat)
        {
            Ouvert.add(solution);
        }
        if (ROOT.getRS().Numsat>maxSat) {
            maxSat= ROOT.Numsat;
        }
        ROOT.getRS().search();
    }
    Remove(ROOT);
}

```

L'algorithme A* :

Environnement de travail :

RAM : 16 Gb

Processeur : Intel Core i5-6600k

Système d'exploitation : Windows 10-64 bits

```
Proc searchAstar() {
    input: heurmode : allow us to choose which heuristic to work with
           nbvar : number of variables
           clauses : Vector containing all the clauses

    output : ouverttemp : a Node object containing a possible
    solution

    Var: int dpth=1;
           int[] instVect = new int[nbvar+1];
           int[] heurVal = calHeur(clauses, nbvar);
           int[] clauseVect = new int[clauses.size()];
           NodeA ouverttemp;
           Vector<NodeA> Ouvert = new Vector<>();

    Start{
createSons(Ouvert, clauses, 0, heurMode, heurVal);
//We create the first nodes of the first depth (depth ==0)
//We then look for the best node in this depth
//After finding it , we iterate on the other depths by creating
    other sons (and choosing the best node at each depth)
    ouverttemp = bestNode(Ouvert); // We look for the best node in Ouvert(
contains all the nodes at a given depth)
    }
}
```

Remarque : Ces pseudo-codes ne résument pas parfaitement le programme implémenté, nous les avons adoptées en fonction des besoins de notre projet, nous avons par exemple, eu besoin du nombre de clauses sat après une certaine limite de temps. Tous ces détails sont bien implémentés dans nos programmes joints avec ce rapport, et s'aperçoivent dans l'interface graphique.

1.2.2. Complexité spatiale et temporelle des algorithmes de résolution

la complexité en temps : le nombre de nœuds créés.

la complexité en mémoire : le nombre maximum de nœuds en mémoire.

Algorithme	Complexité spatiale	Complexité Temporelle
largeur	$O(2^n)$	$O(2^n)$
profondeur	$O(2^n)$	$O(2^n)$
A*	$O(n^2)$	$O(n^2)$

Telle que n = la profondeur.

1.3. Test des algorithmes de résolution

a- Description des données de test:

Nous allons tester les trois algorithmes sur des fichiers benchmark.

Les fichiers utilisés pour les tests sont les benchmarks uf75-325 (des instances satisfiables) et uuf75-325 (des instances non-satisfiables).

Ces benchmarks comportent chacun :

- 100 fichiers en format CNF (100 instances).
- Chaque fichier contient 325 clauses et 75 variables.
- Chaque clause est composée de 3 variables (taille de clause = 3).
- La complexité est en $O(2^{75})$.

Nous allons effectuer nos tests sur 10 instances de chaque fichier.

Nous avons limité le temps d'exécution :

- à 5 minutes pour BFS.
- 1 minutes pour DFS et A*.

et nous avons noté à chaque fois le nombre de clauses SAT.

b- Les résultats obtenues :

Recherche en largeur d'abord :

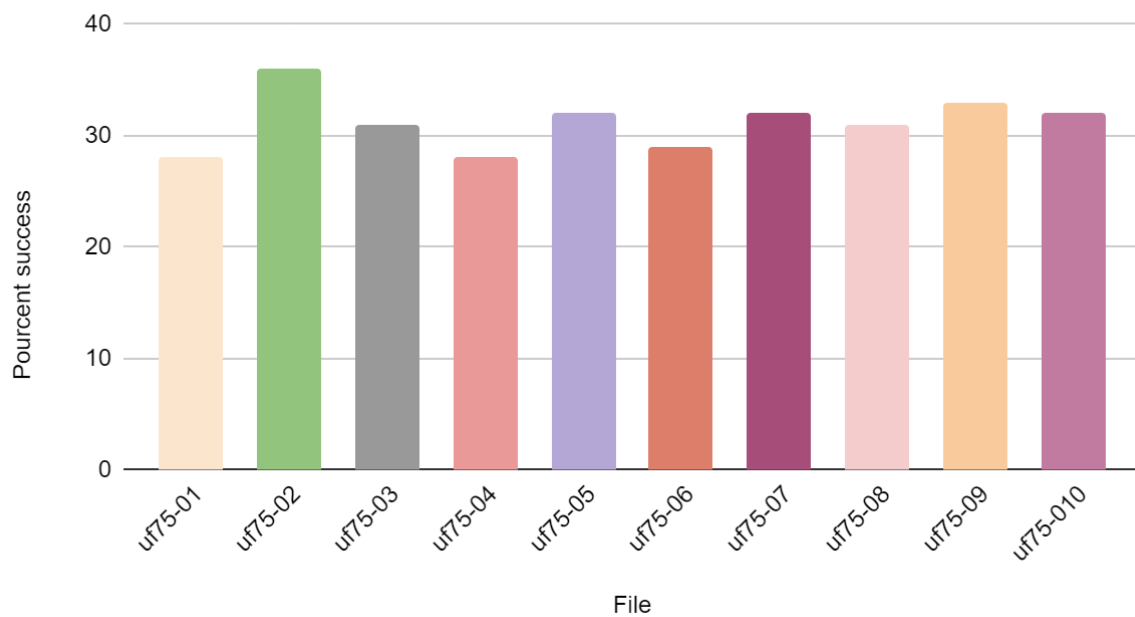
Tableau englobant les résultats :

Instance SAT	Pourcentages des clauses SAT
uf75-01.cnf	28% (93 clauses)
uf75-02.cnf	36% (118 clauses)
uf75-03.cnf	31% (103 clauses)
uf75-04.cnf	28% (92 clauses)
uf75-05.cnf	32% (105 clauses)
uf75-06.cnf	29% (95 clauses)
uf75-07.cnf	32% (106 clauses)
uf75-08.cnf	31% (103 clauses)
uf75-09.cnf	33% (109 clauses)
uf75-010.cnf	32% (104 clauses)

-résultats des instances SAT pour BFS-

Graphe SAT :

Pourcent success par rapport à File



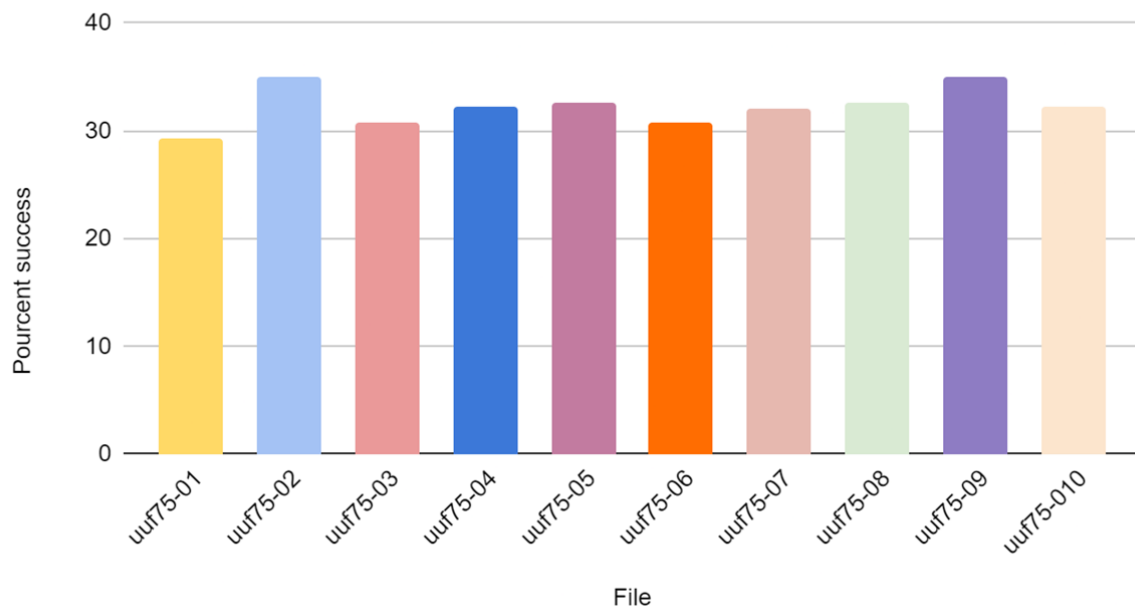
-graphe regroupant les résultats de l'algorithme bfs des fichiers sat-

Instance UNSAT	Pourcentages des clauses SAT
uuf75-01.cnf	29,23% (95 clauses)
uuf75-02.cnf	35,07% (114 clauses)
uuf75-03.cnf	30,76% (100 clauses)
uuf75-04.cnf	32,30% (105 clauses)
uuf75-05.cnf	32,61% (106 clauses)
uuf75-06.cnf	30,76% (100 clauses)
uuf75-07.cnf	32%% (104 clauses)
uuf75-08.cnf	32,61% (106 clauses)
uuf75-09.cnf	35,07% (114 clauses)
uuf75-010.cnf	33,23% (108 clauses)

-résultats des instances NON SAT pour BFS-

Graphe UNSAT:

Pourcent success par rapport à File



-graphe regroupant les résultats de l'algorithme bfs des fichiers unsat-

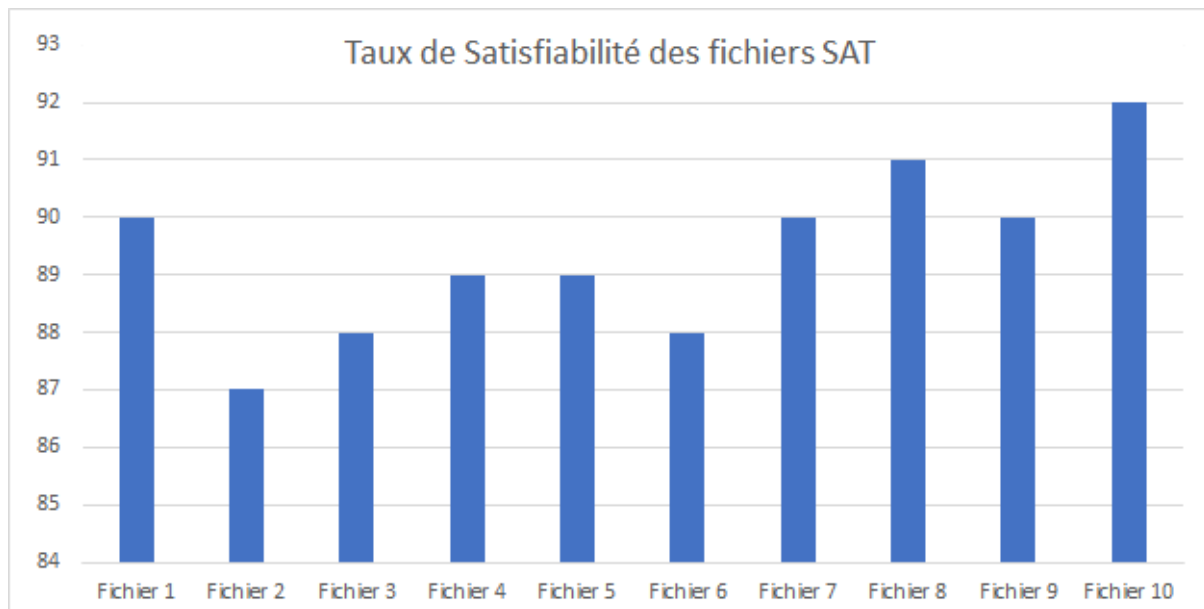
Recherche en profondeur d'abord :

Tableau englobant les résultats SAT:

Instance SAT	Pourcentages des clauses SAT
uf75-01.cnf	90%
uf75-02.cnf	87%
uf75-03.cnf	88%
uf75-04.cnf	89%
uf75-05.cnf	89%
uf75-06.cnf	88%
uf75-07.cnf	90%
uf75-08.cnf	91%
uf75-09.cnf	90%
uf75-10.cnf	92%

-résultats des instances SAT pour DFS-

Graphe SAT :



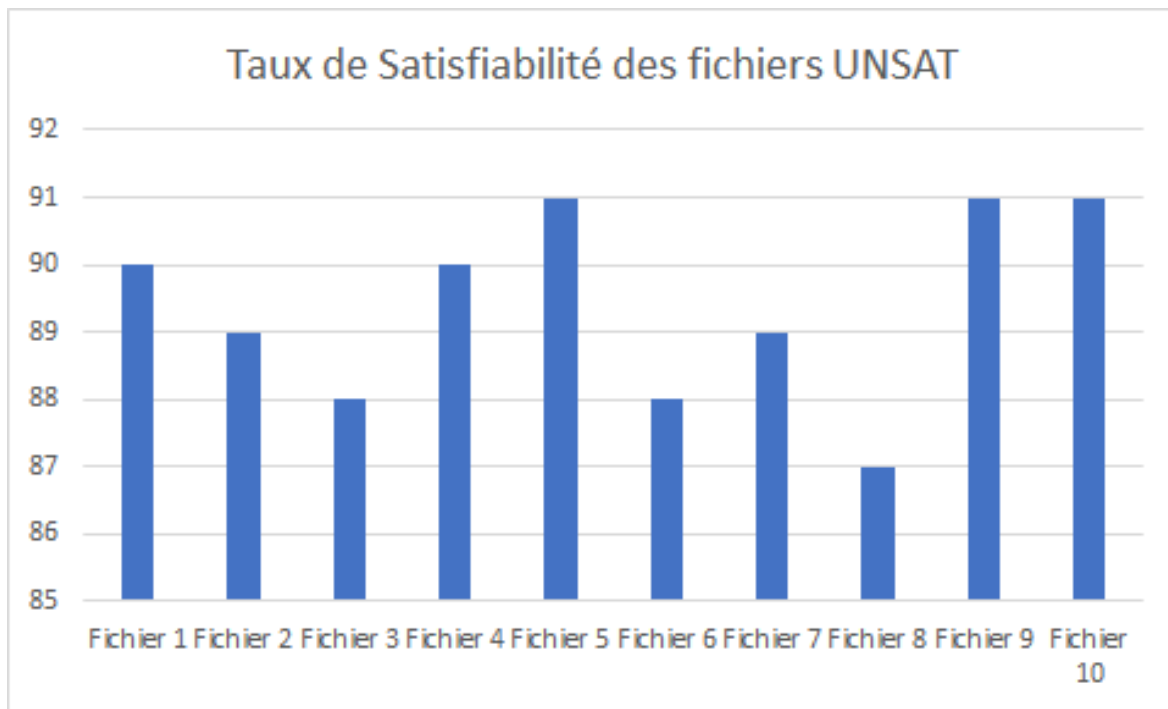
-graphe regroupant les résultats de l'algorithme dfs des fichiers sat-

Tableau englobant les résultats UNSAT:

Instance UNSAT	Pourcentages des clauses SAT
uuf75-01.cnf	90%
uuf75-02.cnf	89%
uuf75-03.cnf	88%
uuf75-04.cnf	90%
uuf75-05.cnf	91%
uuf75-06.cnf	88%
uuf75-07.cnf	89%
uuf75-08.cnf	87%
uuf75-09.cnf	91%
uuf75-010.cnf	91%

-résultats des instances NON SAT pour DFS-

Graphe :



-graphe regroupant les résultats de l'algorithme dfs des fichiers unsat-

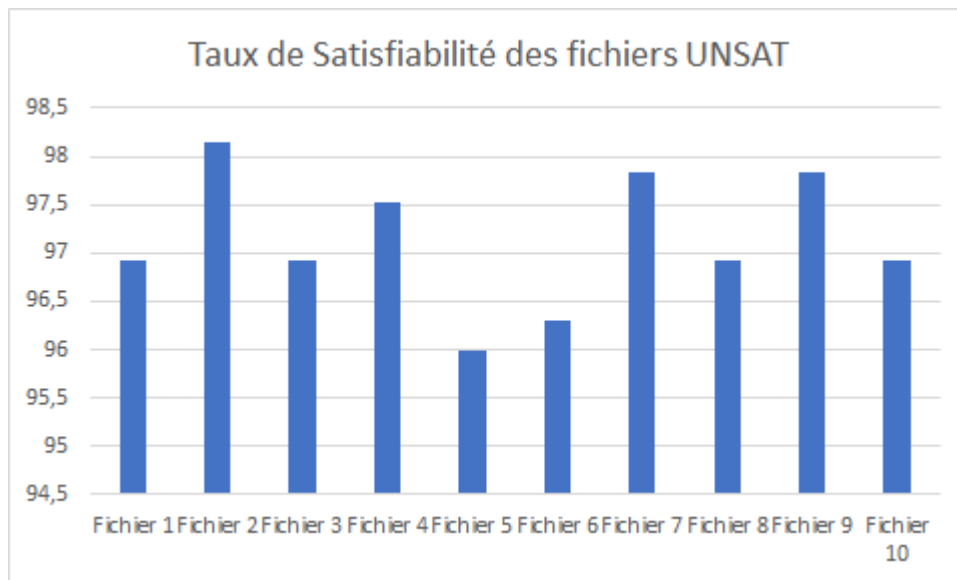
Recherche A* :

Tableau englobant les résultats UNSAT :

Instance NON- SAT	Pourcentages des clauses SAT
uuf75-01.cnf	96.92 % (315 clauses)
uuf75-02.cnf	98.15 % (319 clauses)
uuf75-03.cnf	96.92 % (315 clauses)
uuf75-04.cnf	97.53 % (317 clauses)
uuf75-05.cnf	96 % (312 clauses)
uuf75-06.cnf	96.3 % (313 clauses)
uuf75-07.cnf	97.84 % (318 clauses)
uuf75-08.cnf	96.92 % (315 clauses)
uuf75-09.cnf	97.84 % (318 clauses)
uuf75-010.cnf	96.92 % (315 clauses)

-résultats des instances NON SAT-

Graphe NON-SAT :



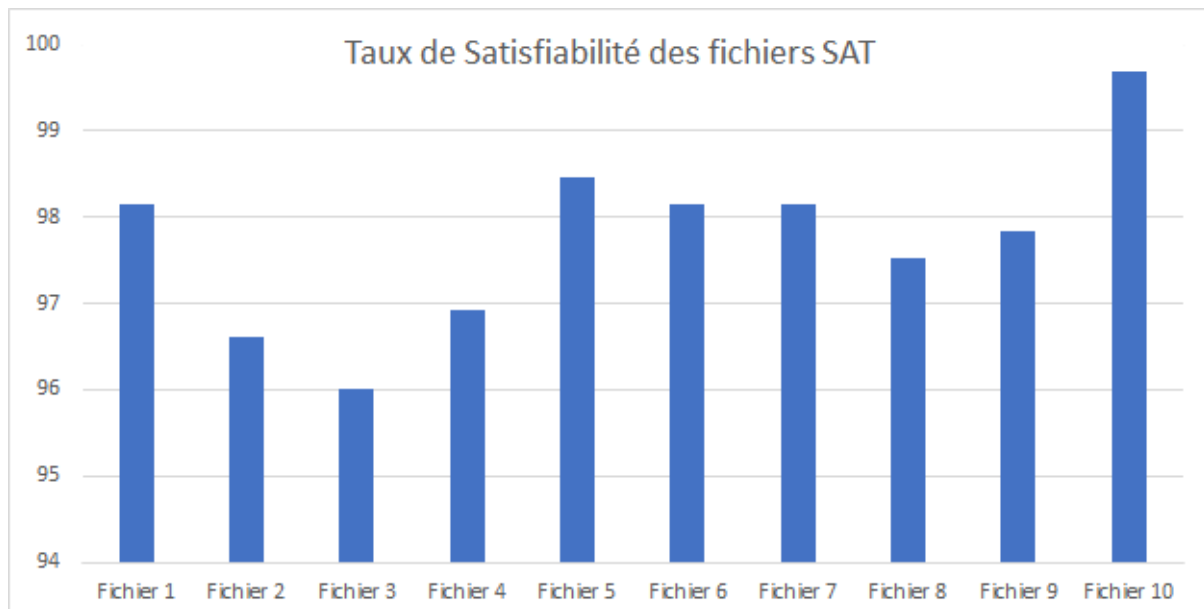
-graphe regroupant les résultats de l'algorithme A* des fichiers unsat-

Tableau englobant les résultats SAT :

Instance SAT	Pourcentages des clauses SAT
uf75-01.cnf	98.15 % (319 clauses)
uf75-02.cnf	96.61 % (314 clauses)
uf75-03.cnf	96 % (312 clauses)
uf75-04.cnf	96.92 % (315 clauses)
uf75-05.cnf	98.46 % (320 clauses)
uf75-06.cnf	98.15 % (319 clauses)
uf75-07.cnf	98.15 % (319 clauses)
uf75-08.cnf	97.53 % (317 clauses)
uf75-09.cnf	97.84 % (318 clauses)
uf75-010.cnf	99.69 % (324 clauses)

-résultats des instances SAT-

Graphe SAT:



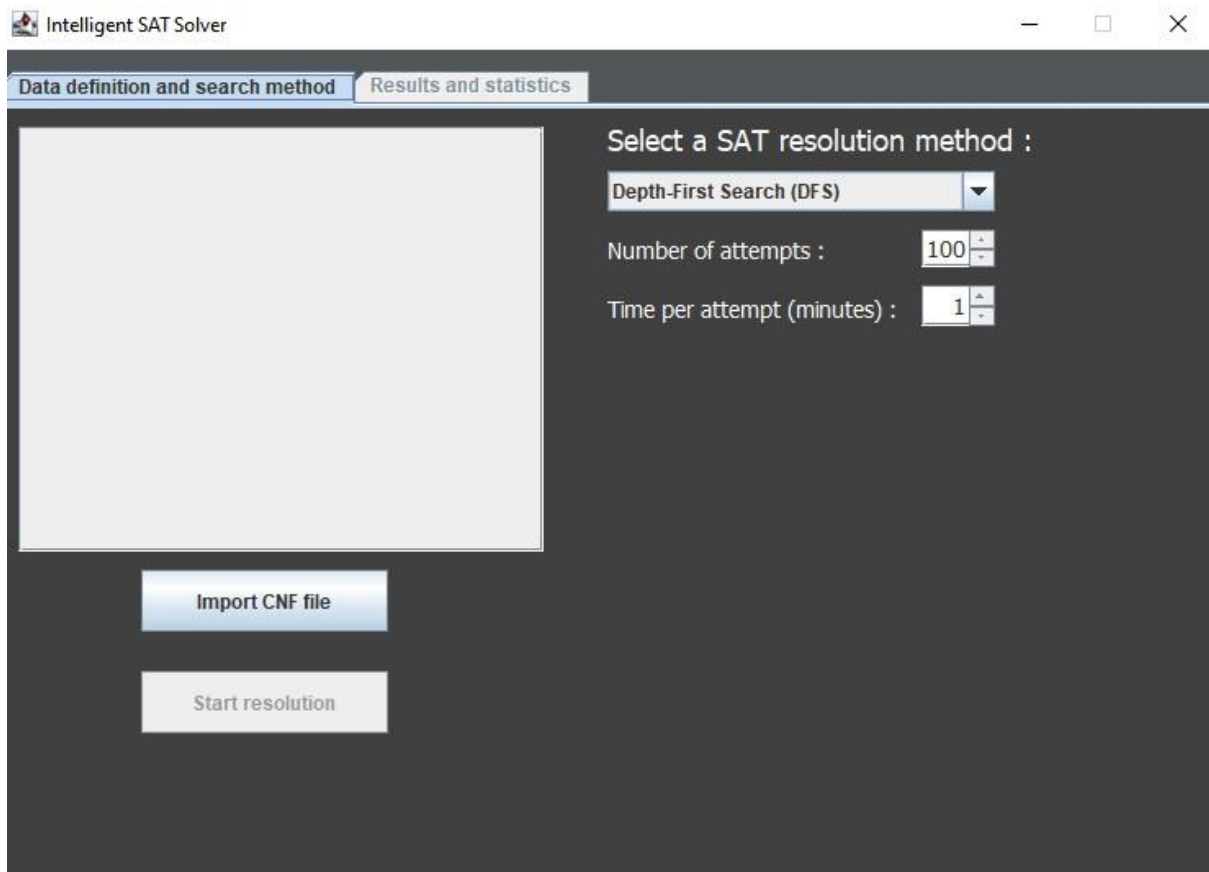
-graphe regroupant les résultats de l'algorithme A* des fichiers unsat-

1.4. Conclusion

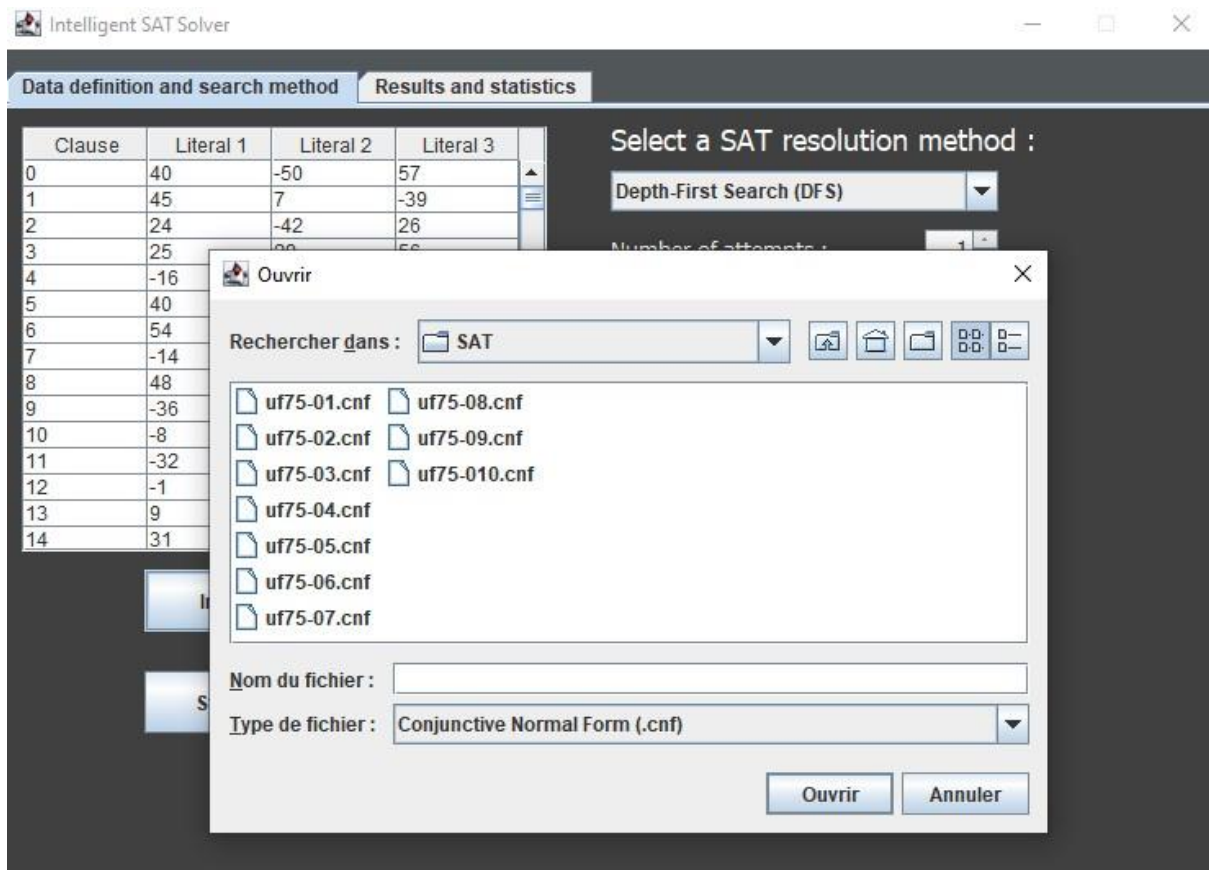
D'après ces résultats , on peut conclure que l'algorithme de recherche A* est beaucoup plus rapide que les algorithmes BFS et DFS , et fournit de meilleurs résultats et ce peu importe l'heuristique.

1.5. INTERFACE GRAPHIQUE

Nous avons conçu une interface graphique afin de faciliter l'exécution de notre programme Java. Elle se présente comme suit :



Une fois le bouton import cnf cliqué, l'utilisateur aura la main pour introduire son fichier,



Il devra ensuite choisir l'algorithme, la limite de temps et le nombre d'essais, cliquer sur start résolution pour afficher les résultats sur la fenêtre 2.

Intelligent SAT Solver

Data definition and search method
Results and statistics

Clause	Literal 1	Literal 2	Literal 3
0	-18	-60	19
1	36	-41	-35
2	-64	75	-19
3	24	-19	22
4	12	72	60
5	-40	42	-68
6	65	-48	-8
7	13	47	-54
8	11	-41	-10
9	-69	12	-54
10	-8	-61	55
11	56	44	31
12	-17	74	-64
13	59	-55	-65
14	-11	66	-56

Import CNF file

Start resolution

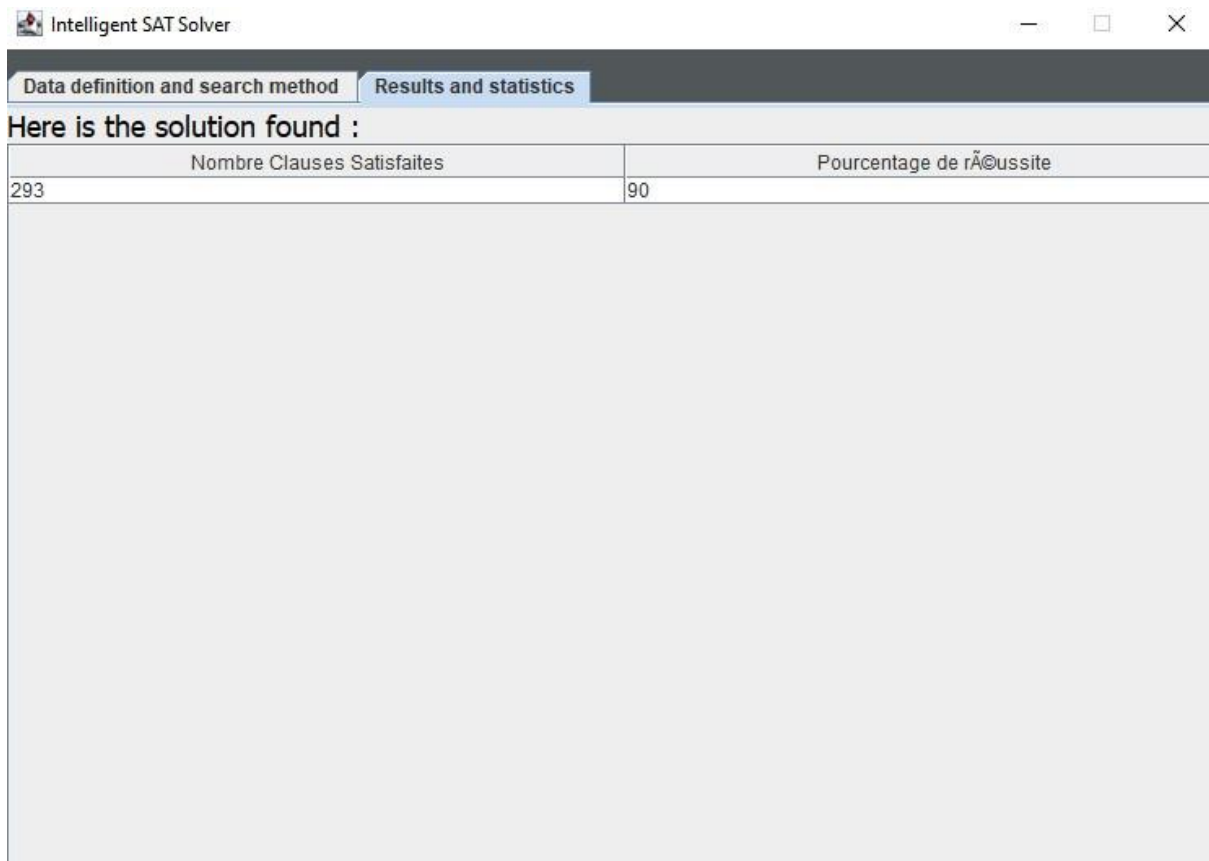
Select a SAT resolution method :

Heuristic Search (A*) First Heuristic

Number of attempts : 100

Time per attempt (minutes) : 1

les résultats :



Nombre Clauses Satisfaites	Pourcentage de réussite
293	90

Dans le cas où on applique l'algorithme A* , le panneau "Results" changera et affichera la solution . Nous aurons alors:

Intelligent SAT Solver		
Data definition and search method		Results and statistics
Here is the solution found :		
-1	-2	3
4	-5	-6
7	-8	-9
10	11	-12
-13	14	15
16	-17	-18
-19	20	21
22	-23	-24
-25	26	27
28	-29	-30
31	32	33
34	-35	-36
-37	-38	39
40	-41	-42
-43	-44	45
-46	-47	48
-49	-50	51
52	53	54
-55	-56	-57
-58	59	60
61	-62	-63
-64	65	66
67	-68	-69
70	-71	72
-73	-74	-75
312	96.0	

Remarque :

Dans le cadre de ce projet , l'instance donnée en solution est divisée de manière à contenir l'instance dans le tableau .

Chaque ligne contient les valeurs de 3 variables de l'instance . Dans le cas de l'image ci-dessus on a:

La première ligne représente les valeurs des variables 1 2 et 3

La seconde ligne représente les valeurs des variables 4 5 et 6

Et cela se poursuit jusqu'au variables 73 74 75

Nous avons également pris la peine d'afficher le pourcentage de succès ainsi que le nombre de clauses satisfaites

Partie II

**-Utilisation des algorithmes
évolutionnaires pour résoudre le
problème SAT-**

Notions de bases²

2.1.1. L'intelligence en essaim (Swarm intelligence)

La Swarm Intelligence ou intelligence distribuée fait référence à la simulation du comportements collectifs d'insectes sociaux comme les fourmis et les abeilles mais aussi les animaux, les poissons et les oiseaux.

Après observation de ces phénomènes intelligents et surtout du comportement de groupe, les chercheurs et plus précisément les psychologues biologiques, ont entrepris des investigations approfondies sur l'interaction sociale de ces agents autonomes lorsqu'ils travaillent ensemble pour atteindre un certain but : chercher la nourriture, construire un nouveau foyer...

Dans le groupe, la communication de l'information influence les comportements de chaque individu et guide tous les agents vers la contribution à l'atteinte d'un même objectif représentant l'objectif de l'essaim. De ce fait, un nouveau paradigme est né sous le nom de 'Swarm Intelligence (SI)' dédié à 'Cooperative Problem Solving (CPS)' -*Résolution de problèmes en coopération*- ces sociétés d'insectes ou d'animaux sont devenus des modèles informatiques., et donc plusieurs méthodes et algorithmes ont été développés s'inspirant de ce nouveau concept.

2.1.2. Méta heuristique, swarm intelligence et algorithmes évolutionnaires

Une heuristique est une connaissance supplémentaire sur le problème à résoudre, utilisée dans un algorithme pour aider à construire une solution plus efficace.

- Il n'est généralement pas facile de le concevoir, car cela dépend du problème.
- Une bonne maîtrise du problème est nécessaire.

Une métaheuristique est une approche qui utilise des heuristiques pour résoudre des problèmes complexes. Il suit un algorithme basé sur une technique spécifique:

- Recherche locale : évolution d'une solution à la fois pour atteindre l'optimum.

² Prof. Habiba Drias Laboratoire de Recherche en Intelligence Artificielle LRIA Computer Science Department USTHB Algiers Algeria, "Swarm Intelligence and Evolutionary Algorithms", 2021.

- Recherche tabou : une recherche locale stratégique (exploitation (bien chercher dans un entourage) et exploration (changer d'endroit de recherche)).

Une approche qui s'inspire de la nature est considérée comme une métaheuristique. Elle suit un schéma d'algorithme évolutif, i.e. inspiré par la simulation de l'évolution naturelle des espèces.

On a donc voulu passer de ces phénomènes naturels vers des algorithmes dit évolutifs basés sur l'intelligence du groupe.

En effet, un algorithme évolutif est un algorithme qui considère une population de solutions de l'espace de recherche et les fait évoluer vers des solutions optimales.

Parmi ces algorithmes évolutionnaires, et dans le cadre de ce tp nous allons étudier l'algorithme génétique et l'algorithme d'optimisation de l'essaim de particules pour résoudre le problème SAT.

2.1. GA : L'algorithme génétique

2.1.1. Principe :

- Les espèces vivantes combattent dans la nature pour survivre. Ceux qui y arrivent sont les plus forts, ils s'accouplent et se reproduisent (notion de **sélection**).
- Ces **enfants** sont similaires (**héritage**), mais pas exactement, que leurs parents à cause du **croisement** et de la **mutation** qui ont eu lieu lors de la production. Ainsi, les enfants peuvent être soit moins ou plus performants que leurs parents.
- Ces enfants reprennent le chemin de leurs parents s'ils survivent.
- Après plusieurs générations, les organismes deviennent beaucoup plus en forme qu'au début (car se sont seulement les meilleures qui survivent).

Sélection naturelle : est un principe qui permet d'optimiser les populations d'individus.

Croisement : reproduction des espèces sélectionnées entre eux.

Mutation : mutation des espèces issues du croisement.

En informatique, l'algorithme évolutionnaire "algorithme génétique" a été développé suivant ces mêmes étapes :

```

Initialiser toute la population
Répéter
    -Sélectionner des individus prometteurs de la population
      actuelle
    -Reproduire les individus sélectionnés
    -Mutation d'individus reproduits
    -Évaluer les individus créés
    -Créer une nouvelle population (mise à jour de la population
      initial) (génération)
Jusqu'à condition d'arrêt
Afficher la meilleure population

```

Les conditions d'arrêt sont multiples, mais en general on veut s'arrêter lorsque la solution est assez proche de la solution la plus optimale :

```

- Lorsqu'on atteint l'objectif défini en entrée.
- X nombre de générations atteintes.
- Lorsqu'il n'y a pas eu d'amélioration de la population après X
  itérations.

```

2.1.2. Avantages :

Dans l'algorithme génétique, a chaque génération plusieurs solutions sont vérifiées à la fois. Ainsi la recherche est assez efficace.

- L'évaluation des individus a chaque génération permet de garder que les meilleures individus (sélection).

2.1.3. Pseudo-algorithme pour résoudre MAX-SAT:

- Etant donné qu'il y a plusieurs méthodes existantes pour ces principaux lignes de l'algorithme, voici comment nous avons travaillé :

Population initial	aléatoire
Sélection	roulette
fonction d'évaluation -fitness-	nombre de clause SAT
Croisement (croiser les individus sélectionner deux à deux)	-prendre deux individus -choisir une position aléatoire dans indiv1 pour construire enfant1

	-choisir une position aléatoire dans indiv2 pour construire enfant2
Mutation (le taux mutation géré par l'utilisateur)	-choisir x nombre aleatoire -muter x variables dans la solution

- Les critères d'arrêt :

Nombre de génération généré = max iterations donné

S'il n'y a pas d'amélioration après un nombre d'itérations (génération) donné

- Pseudo-code :

```

Procédure GenererPopulation
  entrée : n : taille de la population
  sortie : n solutions
  //generer aléatoirement n solutions

Procédure selection
  entrée : n : nombre d'individus à sélectionner
  sortie : n individus sélectionnés
  //selectionner n meilleurs individus

Procédure crossOver
  entrée : 2 individus à croiser
  sortie : 1 enfant
  // 1 individus croisé : une nouvelle solution : croisement des 2

Procédure Mutation :
  entrée : 1 individus à croiser, x : taux de mutation
  sortie : 1 individus muté
  // changer la valeur de x variables

Function Fitness:
  entrée : population, clauses.
  sortie : population avec une évaluation de chaque individu
  //calcul le nombre de clauses sat pour chaque solution

MAIN:
  entrée : Clauses; nbCl;
  PopSize : taille de la population;

```

```

    PourcentageSuccess : pourcentage de succès à atteindre;
    MaxGenerations : nombre maximal d'itérations;
    MaxGenStop : nombres de générations maximale s'il ne
                  s'améliore pas ;
    TauxMut : taux de mutation;

sortie : solution la plus optimale

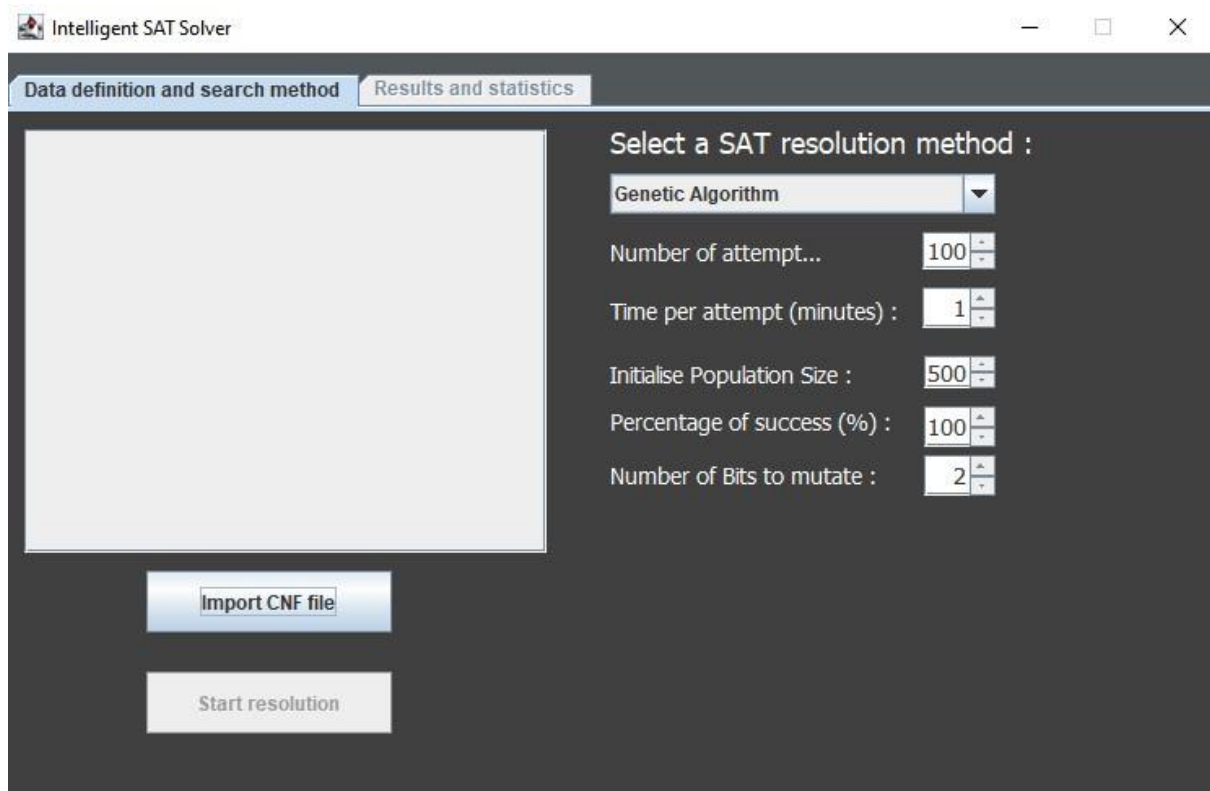
var :
    list population, selectedpop, childs,generation;
    int nbvar;
    int solution;
begin
    population = GenererPopulation(clauses, nbvar, PopSize, nbcl);
    fitness(population);
    selectedpop=selection(population);
    while(selectedpop != vide)
    faire
        pour indiv1,indiv2 de selectedpop
        faire
            childs=crossover(indiv1, indiv2);
            mutate(child[1],TauxMut);
            mutate(child[2],TauxMut);
            fitness(childs);
            generation.add(childs)
        fait;
        solution=bestSolution;
    fait
    afficher(solution);
end.

```

2.1.4. Mise à jour de l'interface:

Afin que l'interface colle le mieux aux spécificités de cet algorithme , nous avons ajouté des boutons permettant d'initialiser les différents paramètres , tels que la taille de la population ou le pourcentage de succès à atteindre

Pour ce qui est de l'affichage du résultat , le logiciel affiche l'instance qu'il a trouvé dans la fenêtre "result and statistics" .



2.1.5. Tests et résultats :

Nous avons fait varier les paramètres d'entrées pour tester les fichiers benchmark (10 instances des fichiers sat et 10 sur unsat) le but est toujours d'atteindre une solution de 100%, et l'algorithme s'arrête s'il ne s'améliore pas au bout de générationMax = 500 générations.

Les paramètres et les résultats sont résumés dans les tableaux suivants :

	1	2	3	4	5
taille de la population	500	800	1000	1200	1500
nombre d'itérations	800	900	1000	1500	2000
nombre de gènes à mutés	50	70	80	90	100

tableau regroupant les différents paramètres de test

Instance SAT	1	2	3	4	5
--------------	---	---	---	---	---

uf75-01.cnf	323 clauses (99,38%)	323 clauses (99,38%)	324 clauses (99.69%)	324 clauses (99.69%)	324 clauses (99.69%)
uf75-02.cnf	319 clauses (98,15%)	322 clauses (99,07%)	324 clauses (99.69%)	324 clauses (99.69%)	324 clauses (99%)
uf75-03.cnf	319 clauses (98,15%)	319 clauses (98,15%)	323 clauses (99,38%)	319 clauses (98.15%)	322 clauses (99.07%)
uf75-04.cnf	320 clauses (98,46%)	322 clauses (99,07%)	322 clauses (99.07%)	321 clauses (98.76%)	323 clauses (99.38%)
uf75-05.cnf	318 clauses (97,84%)	321 clauses (98,76%)	321 clauses (98,76%)	322 clauses (99,08%)	322 clauses (99.07%)
uf75-06.cnf	321 clauses (98,76%)	321 clauses (98,76%)	321 clauses (98,76%)	323 clauses (99,38%)	322 clauses (99.07%)
uf75-07.cnf	321 clauses (98,76%)	321 clauses (98,76%)	322 clauses (99,07%)	322 clauses (99,07%)	322 clauses (99.07%)
uf75-08.cnf	322 clauses (99,07%)	322 clauses (99,07%)	322 clauses (99,07%)	322 clauses (99,07%)	322 clauses (99.07%)
uf75-09.cnf	323 clauses (99,38%)	324 (99,69%)	323 clauses (99,38%)	324 clauses (99,69%)	325 clauses (100%)
uf75-010.cnf	321 clauses (98,76%)	324 (99,69%)	324 (99,69%)	324 clauses (99,69%)	325 clauses (100%)

Tableau englobant les résultats des fichiers SAT

Instance UNSAT	1	2	3	4	5
uuf75-01.cnf	322 clauses (99,07%)	322 clauses (99,07%)	323 clauses (99,38%)	321 clauses (98,77%)	322 clauses (99,08%)
uuf75-02.cnf	319 clauses (98,15%)	322 clauses (99,07%)	324 (99,69%)	322 clauses (99,08%)	323 clauses (99,38%)
uuf75-03.cnf	321 clauses (98,76%)	322 clauses (99,07%)	321 clauses (98,76%)	321 clauses (98,77%)	322 clauses (99,08%)
uuf75-04.cnf	316 clauses (97,23%)	322 clauses (99,07%)	320 clauses (98,46%)	321 clauses (98,77%)	321 clauses (98,77%)
uuf75-05.cnf	319 clauses (98,15%)	320 clauses (98,46%)	320 clauses (98,46%)	322 clauses (99,08%)	322 clauses (99,08%)

uuf75-06.cnf	320 clauses (98,46%)	321 clauses (98,76%)	320 clauses (98,46%)	320 clauses (98,46%)	323 clauses (99,38%)
uuf75-07.cnf	319 clauses (98,15%)	321 clauses (98,76%)	321 clauses (98,76%)	320 clauses (98,46%)	321 clauses (98,77%)
uuf75-08.cnf	321 clauses (98,76%)	321 clauses (98,76%)	323 clauses (99,38%)	323 clauses (99,38%)	323 clauses (99,38%)
uuf75-09.cnf	322 clauses (99,07%)	324 (99,69%)	323 clauses (99,38%)	322 clauses (99,08%)	323 clauses (99,38%)
uuf75-010.cnf	321 clauses (98,76%)	321 clauses (98,76%)	322 clauses (99,07%)	320 clauses (98,46%)	322 clauses (99,08%)

Tableau englobant les résultats des fichiers UNSAT

Discussion et conclusion

Nous remarquons que la taille de la population est proportionnelle à la qualité de la solution . On obtient plus souvent d'excellentes solutions (voir parfaites) en augmentant la taille de la population. Par exemple , lorsque la taille de la population = 1500 , nous avons obtenu dans deux fichiers différents 100% de clauses satisfaites.

2.2. PSO : Algorithme d'optimisation de l'essaim de particules ³

2.2.1. Principe :

L'algorithme d'optimisation de l'essaim est un algorithme évolutionnaire basé sur la recherche d'une solution optimale dans l'espace de recherche.

C'est une technique simulant l'évolution du mouvement naturel et non l'évolution biologique tel que GA, donc la notion de sélection n'existe pas.

Il a été inspiré à partir de l'observation de :

- Volées d'oiseaux à la recherche de nourriture.
- Bancs de poissons évitant les prédateurs.
- Tout comportement similaire basé sur l'interaction sociale.

```
Initialize N particles: positions and velocities;
Evaluate the particles positions;
for each particle i do
    Pbesti = xi ;
    calculate Gbest using the transition rule;
    for each iteration do
        for each particle p do
            update the velocity and the position;
            move the particle and evaluate its fitness;
            update Pbest;
        endfor;
        update Gbest;
    endfor;
end
```

2.2.2. Avantages :

- Mise en œuvre simple et courte.
- Équilibre entre exploration et exploitation
- Utile pour la vie artificielle.
- Efficace pour optimiser plusieurs fonctions.

³ Prof. Habiba Drias Laboratoire de Recherche en Intelligence Artificielle LRIA Computer Science Department USTHB Algiers Algeria, "Particle Swarm Optimization PSO", 2021.

Malgré tous ces avantages, PSO manque d'efficacité.

2.2.3. Pseudo-algorithme:

```
Fonction GenSwarm(Vecteur de clauses, nbvariables,nbclauses,taille
essaim) : // Initialiser l'essaim

Procédure initpBest(Vecteur de Particules représentant l'essaim):
//initialiser le pBest de chaque particule

Procédure FitnessCal(Particule, Vecteur de clauses):
// Calculer le score de fitness d'une particule donnée

Fonction calculation(Vecteur de particules):
//Rend le meilleur pBest du vecteur de particule

Procédure updateVelocity(Particule p, poids d'inertie,
const1,const2,gBest,pBest):
//Met à jour la vitesse d'une particule donnée

Procédure updatePosition(Particule p):
//Mets à jour la position d'une particule donnée

Fonction distance(Particule p1,Particule p2):
//Retourne la distance de Hamming entre deux position

Procédure void updatePbest(Particule p):
//Mets à jour la pBest d'une particule donnée

entrée : const1 c1, const2 c2, poids d'inertie w, nombre
itérations iterMax
sortie : un objet SAT représentant la meilleure solution
begin
pour(i< maxIter)
faire
    (pour chaque particule de l'essaim
    faire
        //mettre à jour la vitesse
        //mettre à jour les fitness
        //mettre à jour la position
```

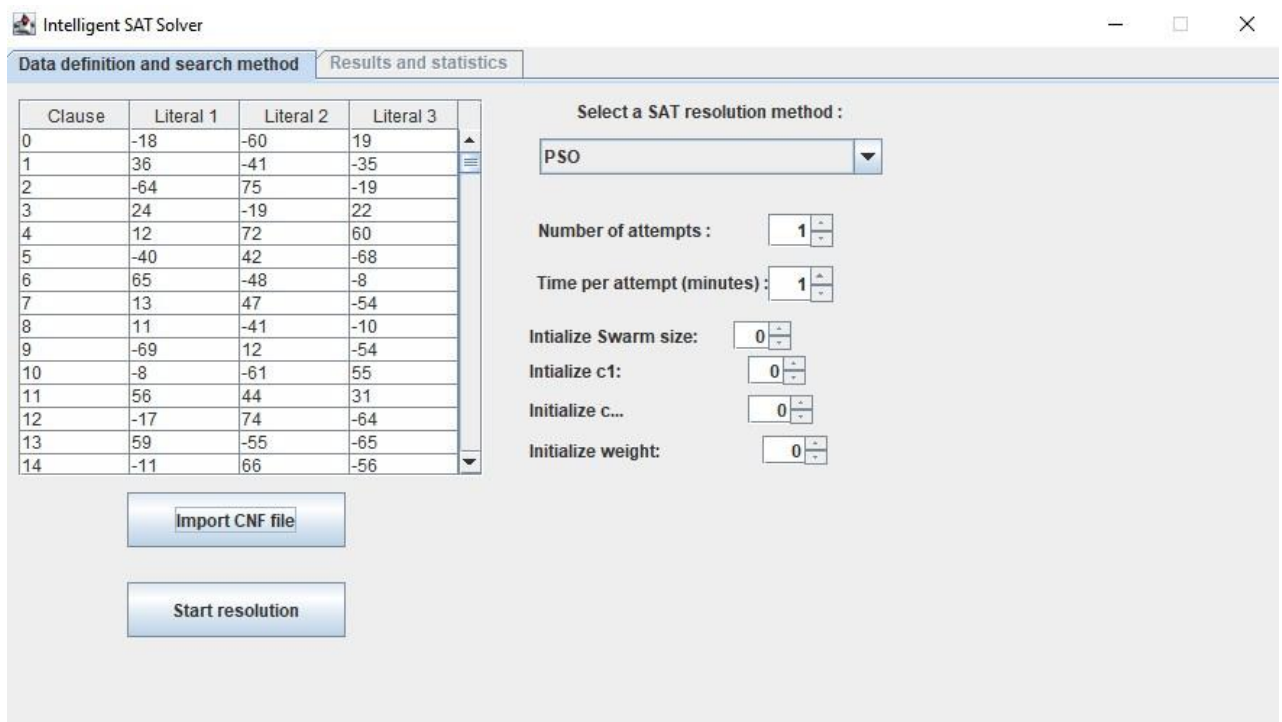
```

//mettre à jour la pBest (la meilleure solution de chaque
particule)
fait;)
mettre à jour le gbest (la meilleure solution)
fait;
end.

```

2.1.4. Mise à jour de l'interface:

Afin que l'interface colle le mieux aux spécificités de cet algorithme , nous avons ajouté des boutons permettant d'initialiser les différents paramètres , tels que la taille de l'essaim ou encore le poids d'inertie.



2.2.5. Tests et résultats :

Nous avons fait varier les paramètres d'entrées pour tester les fichiers benchmark (10 instances des fichiers sat et 10 sur unsat) les paramètres et les résultats sont résumés dans les tableaux suivants :

	1	2	3	4	5
--	---	---	---	---	---

taille de l'essai	1000	1200	1400	1500	1700
nombre d'itérations max	100	250	300	700	700
constante c1	2	1	0	1	2
constante c2	2	2	1	2	1
Poids d'inertie w	200	300	400	500	1000

Tableau regroupant les différents paramètres de test

Instance SAT	1	2	3	4	5
uf75-01.cnf	307 (94,46%)	308 (94,76%)	309 (95,07%)	310 (95,38%)	311 (95,69%)
uf75-02.cnf	309 (95,07%)	311 (95,69%)	310 (95,38%)	312 (96,00%)	313 (96,30%)
uf75-03.cnf	306 (94,15%)	307 (94,46%)	309 (95,07%)	311 (95,69%)	310 (95,38%)
uf75-04.cnf	309 (95,07%)	308 (94,76%)	308 (94,76%)	309 (95,08%)	311 (95,69%)
uf75-05.cnf	311 (95,69%)	310 (95,38%)	312 (96%)	313 (96,31%)	311 (95,69%)
uf75-06.cnf	308 (94,76%)	309 (95,07%)	310 (95,38%)	313 (96,31%)	311 (95,69%)
uf75-07.cnf	309 (95,07%)	309 (95,07%)	309 (95,07%)	309 (95,08%)	311 (95,69%)
uf75-08.cnf	311 (95,69%)	310 (95,38%)	310 (95,38%)	309 (95,08%)	311 (95,69%)
uf75-09.cnf	310 (95,38%)	309 (95,07%)	309 (95,07%)	311 (95,69%)	311 (95,69%)
uf75-010.cnf	310 (95,38%)	309 (95,07%)	309 (95,07%)	310 (95,38%)	312 (96%)

Tableau englobant les résultats des fichiers SAT

Instance UNSAT	1	2	3	4	5
uuf75-01.cnf	308 (94,76%)	308 (94,76%)	308 (94,76%)	308 (94,77%)	311 (95,69%)
uuf75-02.cnf	308 (94,76%)	311 (95,69%)	309 (95,07%)	311 (95,69%)	311 (95,69%)
uuf75-03.cnf	306 (94,15%)	308 (94,76%)	308 (94,76%)	311 (95,69%)	311 (95,69%)
uuf75-04.cnf	306 (94,15%)	307 (94,46%)	308 (94,76%)	308 (94,77 %)	310 (95,38%)
uuf75-05.cnf	305 (93,84%)	310 (95,38%)	309 (95,07%)	309 (95,08%)	310 (95,38%)
uuf75-06.cnf	308 (94,76%)	308 (94,76%)	308 (94,76%)	308 (94,77 %)	308 (94,76%)
uuf75-07.cnf	306 (94,15%)	307 (94,46%)	309 (95,07%)	311 (95,69%)	310 (95,38%)
uuf75-08.cnf	307 (94,46%)	308 (94,76%)	308 (94,76%)	309 (95,07%)	311 (95,69%)
uuf75-09.cnf	309 (95,07%)	310 (95,38%)	309 (95,07%)	310 (95,38%)	311 (95,69%)
uuf75-010.cnf	306 (94,15%)	309 (95,07%)	308 (94,76%)	310 (95,38%)	308 (94,76%)

Tableau englobant les résultats des fichiers UNSAT

Discussion et conclusion

Nous remarquons une certaine stagnation dans les résultats (nous dépassons rarement les 95%) . En effet , le pourcentage des clauses satisfaites est grandement influencé par les constantes initialisées (c1, c2 et surtout le w) et rarement par la taille de l'essaim.

Il est fort probable qu'en augmentant ces valeurs nous obtenons de meilleurs résultats.