Faculté des sciences

République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université Ben Youcef Benkhedda Alger 1

Département mathématiques et informatique

Mini projet

3-SAT Problem Resolving

Bio inspiré TP

Réaliser par :

- Bala Abdelkarim
- Mehadjbia Abdelhak
- Addad Malek

- Team 18
- Year 2021/2020
- Class ISII master 2

Sommaire

Introduction

I) Introduction

1) Présentation de problème 3-SAT

Le problème SAT est un problème de logique propositionnelle, encore appelée logique d'ordre 0 ou logique des prédicats. Rappelle :

- Un prédicat est défini sur ensemble de variables logiques, à l'aide des 3 opérations élémentaires que sont la négation (¬x que nous noterons aussi $\bar{x}$),la conjonction ET (x∧y) et la disjonction OU (x∨y).
- Un littéral est un prédicat formé d'une seule variable(x) ou de sa négation $\bar{x}$.
- Une clause est formée uniquement de la disjonction de littéraux
- 3-sat représente un problème SAT tel que le nombre de littéraux est 3 pour chaque clause.

(i) Définition de data set 3-SAT

Lien de data set	Nombre d'instances	Nombre des littéraux	Nombre des clauses
http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html	100	75	325

Voici un exemple démonstratif :

```
42 22 15 0  ----->   x42 ^ x22 ^ x15 = c1
73 -22 -24 0 ----->  x73 ^ ¬x22 ^ ¬x24 = c2
50 -20 53 0  ----->  x50 ^ ¬x20 ^ x53 = c3
```

Notre objective est de choisir la meilleure instanciation des littéraux pour rendre tout l'instance satisfait, et pour faire ça nous avons utilisé différent algorithmes comme A étoile, recherche en profondeur, algorithme génétique et colonie de fourmi.

2) La structure globale de rapport

Dans la prochain partie on va voir une solution proposée par nous pour résoudre ce type de problème, notons que nous avons travaillé sur une solution déjà existé et qu'il est réalisé par des étudiants de la promo de l'année précédente avec des modification et une amélioration sur la structure globale de projet.

Pour organisé bien notre travaillé nous avons le divisé en deux étape important :

1. **La codification de problème :** dans cette étape on donne la modélisation générale de problème comme la définition des structures des données utilisé.
2. **Les algorithmes utilisés :** cette étape est dédié à l'implémentation des algorithmes qui permettent de calculé ou trouvé une solution à ce problème, ainsi on voir la performance de chaque algorithme et faire une comparaison entre ces algorithmes de recherche
3. **La conclusion :** on donne le meilleur algorithme qui s'adapte avec le problème 3-SAT, avec des avantages et des inconvénients des différents algorithmes utilisés.

II) Pratique

1) Codification de problème

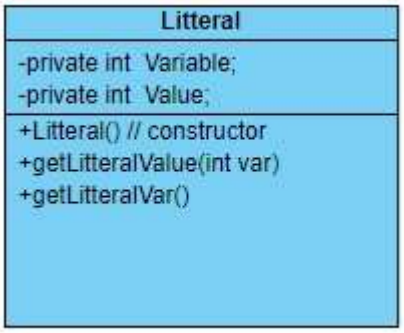
Pour la modélisation de ce problème nous avons utilisé le paradigme orienté objet et comme une initialisation nous avons traces le diagramme de class dédié à la représentation de problème 3SAT.

On distingue 3 class générale qui décrivant le problème SAT :

1. **La class littéral :** est la class le plus simple et le plus bas dans la hiérarchique de problème SAT, il contient comme attribues deux variables, **Variable** qui représente le nom de littéral, l'intervalle de valeur qui le peut prendre est entre [1 et 75], et **Value** qui représente la valeur réel de l'instanciation, il est soit 0 s'il n'existe pas ou 1 s'il existe ou -1 s'il est en état de négation.

Comme des méthodes on trouve :

- Les getters et les setters des deux variables (**Variable, Value**).

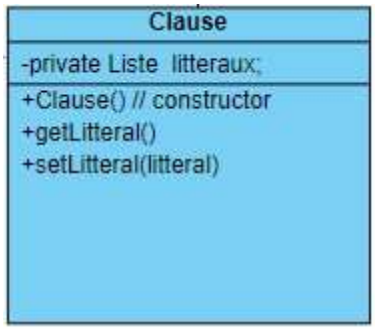


2. **La class clause :** représente la conjonction des littéraux, il contient un attribut **littéraux** qui

est un liste des littéral.

Comme des méthodes on trouve :

- getLatteral et setLatteral (getters and setter)

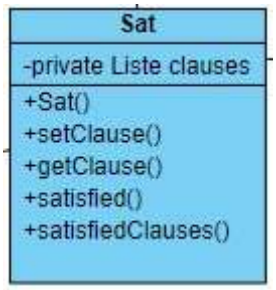


3. **La class sat :** représente l'instance de problème SAT, il contient un attribut **clauses** qui est un liste des clauses, pour dire que le problème est résolu il faut que tous les éléments de liste **clauses** ont la valeur 1.

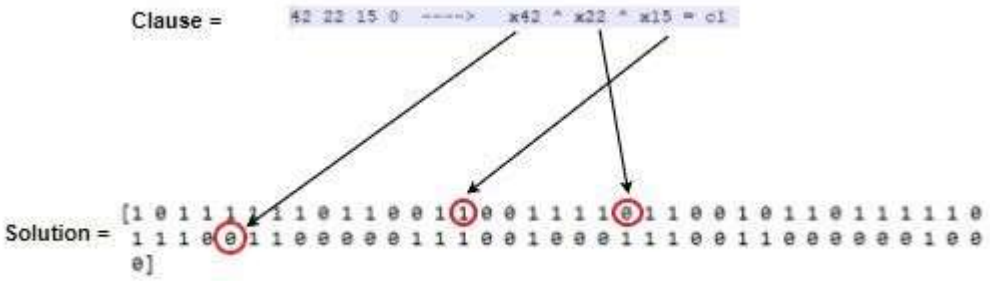
Comme des méthodes on trouve :

- les setters et les getters.

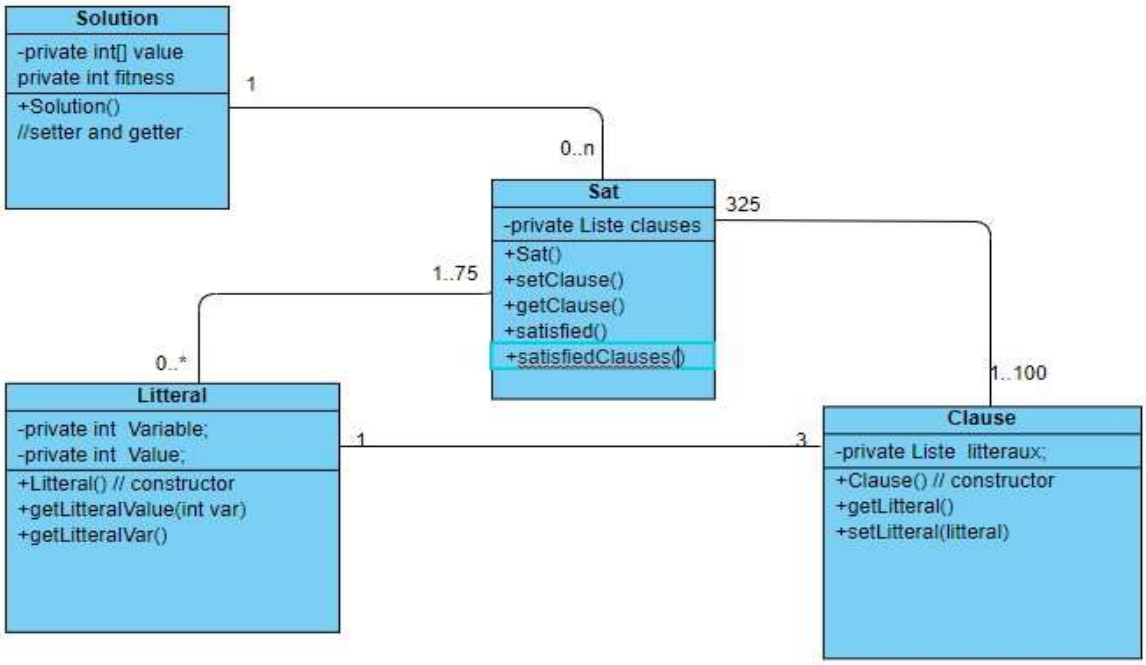
- Satisfied (solution) retourner vrai si tous les clauses de instanciation est 1 sinon retourner faux.
- satifiedClause (solution) retourner le nombre des clauses satisfaits c à d qu'ils ont la valeur 1.



4. **La class Solution :** représente la solution qu'il faut le trouver pour rendre l'instance satisfait, dans tous les algorithmes il faut faire l'initialisation de solution de façon quelque conque ou à zéro ou à -1 par apport au algorithme. La class contient un tableau des entiers qui prendre la valeur 0, 1, ou -1, et d'autre attribues comme fitness, livel…etc.



(i) diagramme de class globale

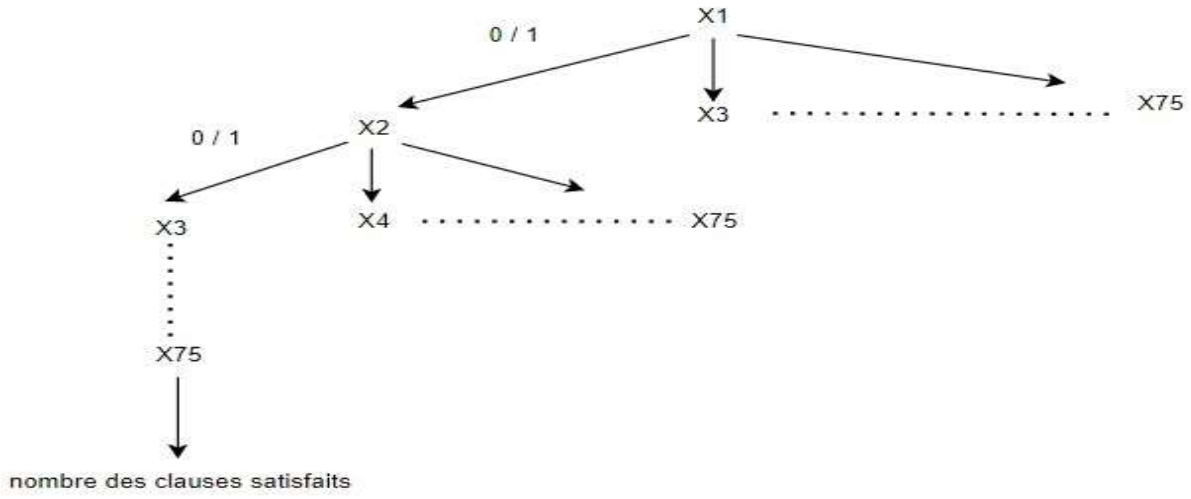


2) Les algorithmes utilisés

Pour résoudre notre problème 3-SAT, nous avons implémenté 3 types de recherche de solution, non heuristique comme recherche en profondeur, heuristique comme A étoile et Meta heuristique comme l'algorithme génétique et Ant colony optimisation qu'ils sont bio inspiré.

(i) Depth First Search Algorithm (DFS)

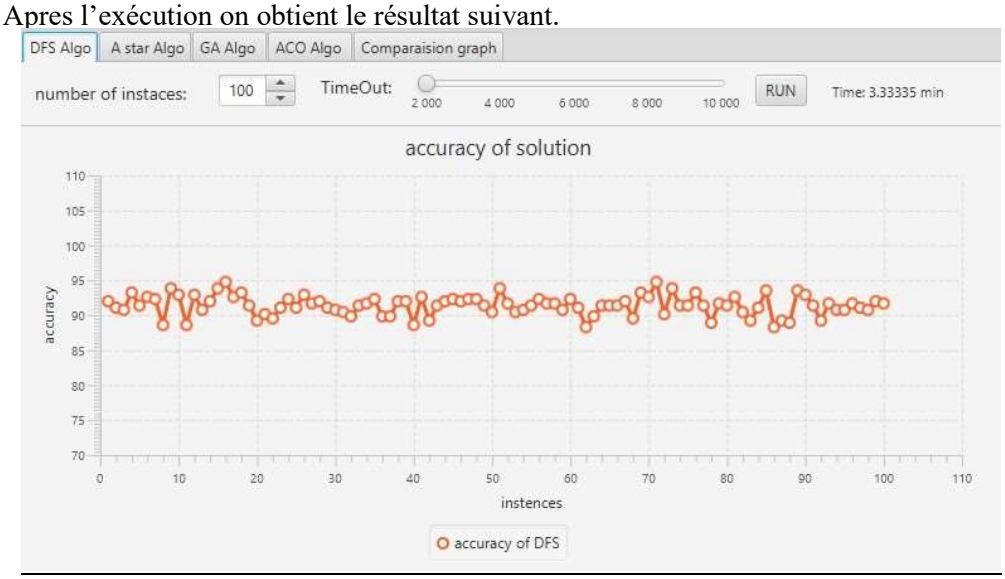
La recherche en profondeur traverse en explorant chaque chemin le plus loin possible avant de devoir revenir en arrière. C'est la raison pour laquelle vous pouvez également trouver cet algorithme sous le nom de **Backtracking (Retour en arrière)**.
Dans notre cas, à partir d'un état initial d'un littéral on prendre tous les combinaisons possible des instanciations des littéraux restant jusqu'à ou on trouve le meilleur chemin

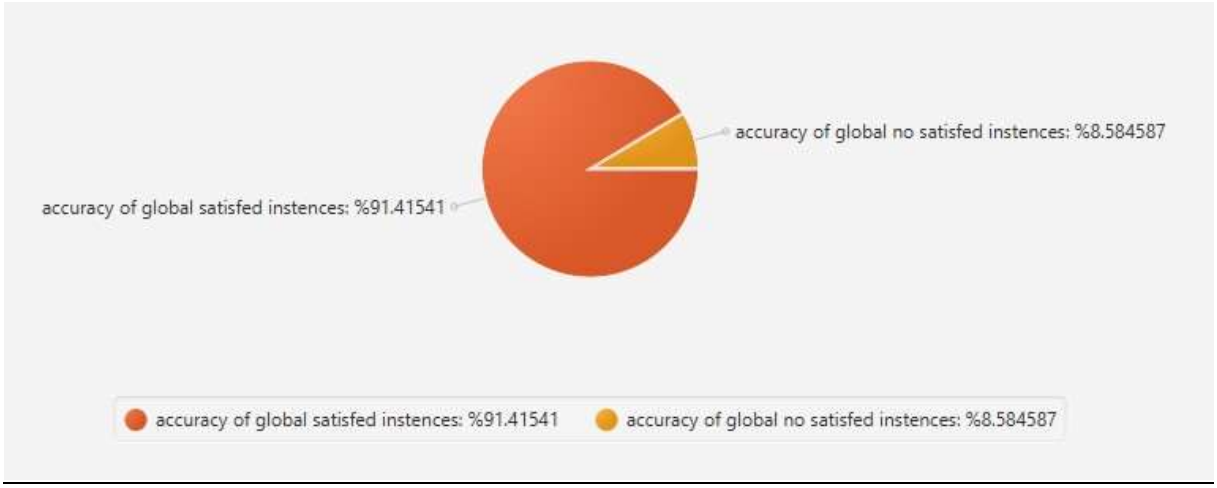


- <u>Les paramètres utilisés dans l'algorithme</u>
  Dans cet algorithme nous avons utilisé deux paramètres, **TimeOut** pour éviter de tombé dans des cas ou l'algorithme boucle d'une manière infinie où il prendre beaucoup de temps pour trouver une solution.
  Le deuxième paramètre est le nombre d'instances qu'il doit résolu.

- <u>L'exécution de l'algorithme</u>

Apres l'exécution on obtient le résultat suivant.





(ii) A star Algorithm

L'algorithme A* est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final c a d le nœud qui a un fitness le plus grand. Il utilise une évaluation heuristique sur chaque nœud pour estimer le meilleur chemin y passant, et visite ensuite les nœuds par ordre de cette évaluation heuristique. C'est un algorithme simple.

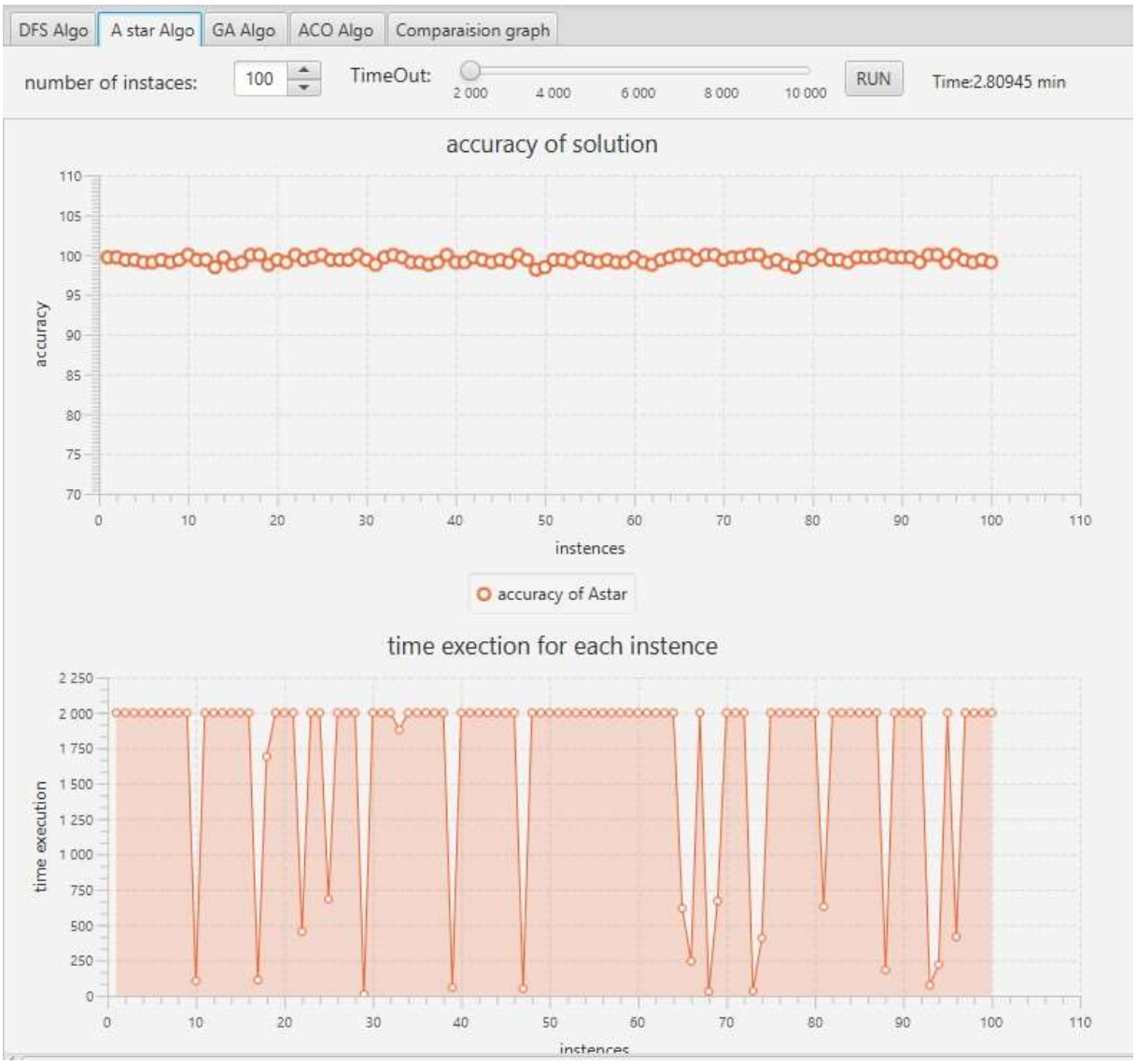
- Les paramètres utilisés dans l'algorithme

  Dans cet algorithme nous avons utilisé deux paramètres, **TimeOut** pour éviter de tombé dans des cas ou l'algorithme boucle d'une manière infinie où il prendre beaucoup de temps pour trouver une solution.

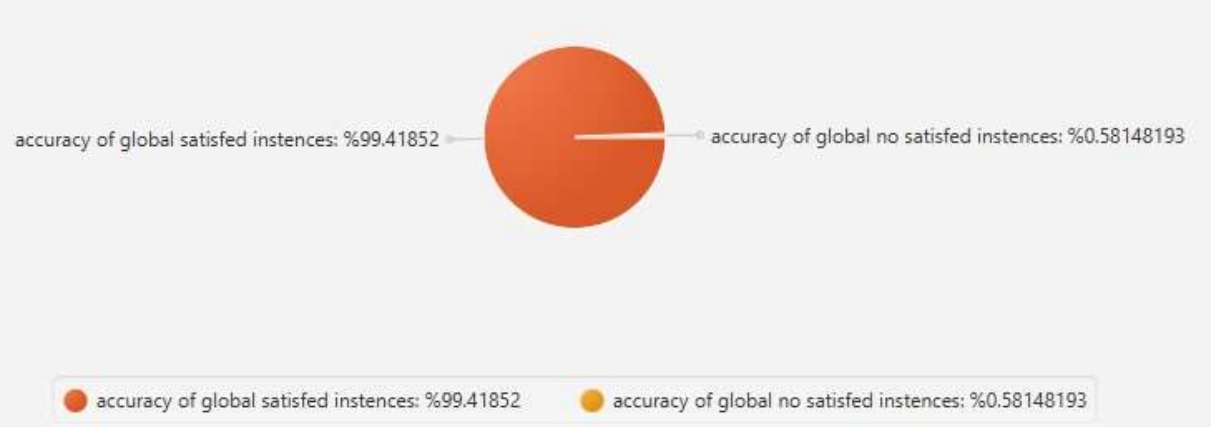
  Le deuxième paramètre est le nombre d'instances qu'il doit résolu.

- L'exécution de l'algorithme

  Apres l'exécution on obtient le résultat suivant.

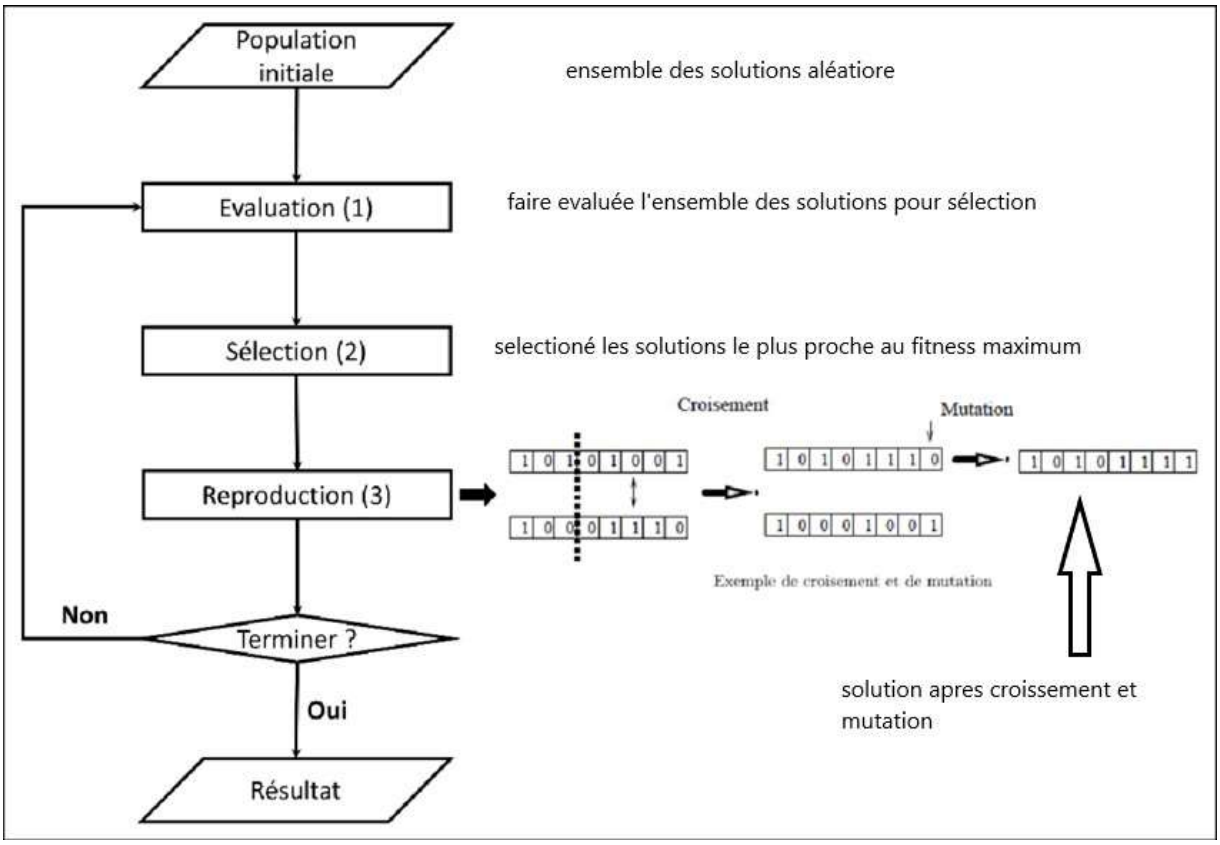


- Le pourcentage de satisfaction de toutes les clauses on utilise l'algorithme A-étoile.



(i) Genetic Algorithm (**GA**)

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes. Leur but est d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode exacte (ou que la solution est inconnue) pour le résoudre en un temps raisonnable. Les algorithmes génétiques utilisent la notion de sélection naturelle et l'appliquent à une population de solutions potentielles au problème donné. La solution est approchée par « bonds » successifs, comme dans une procédure de séparation et évaluation (branch & bound), à ceci près que ce sont des formules qui sont recherchées et non plus directement des valeurs [Wikipédia]. ▪ Procédure de travaillé :

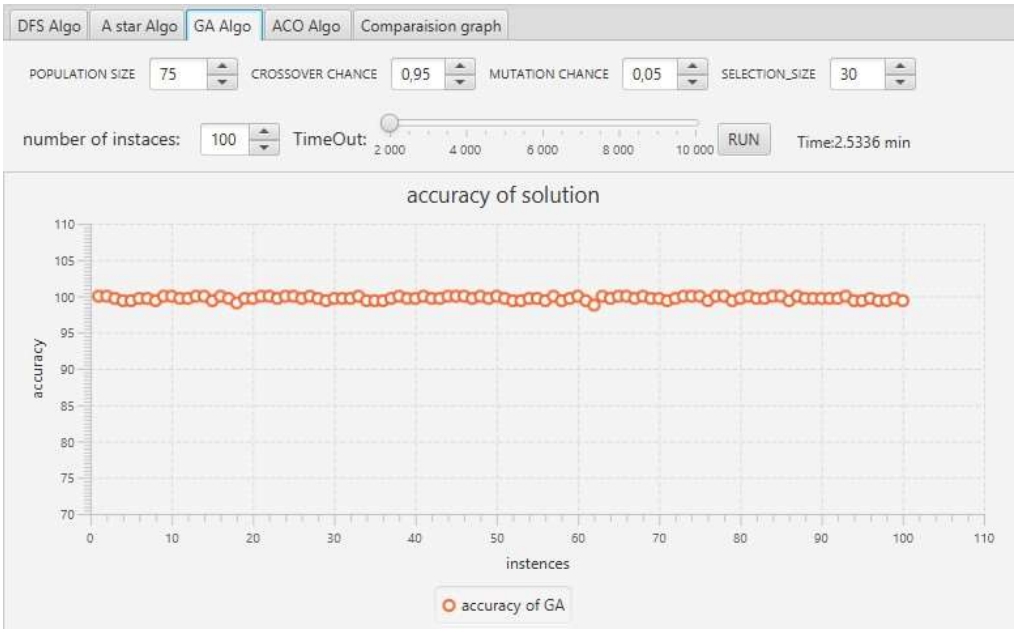


- Les paramètres utilisés dans l'algorithme

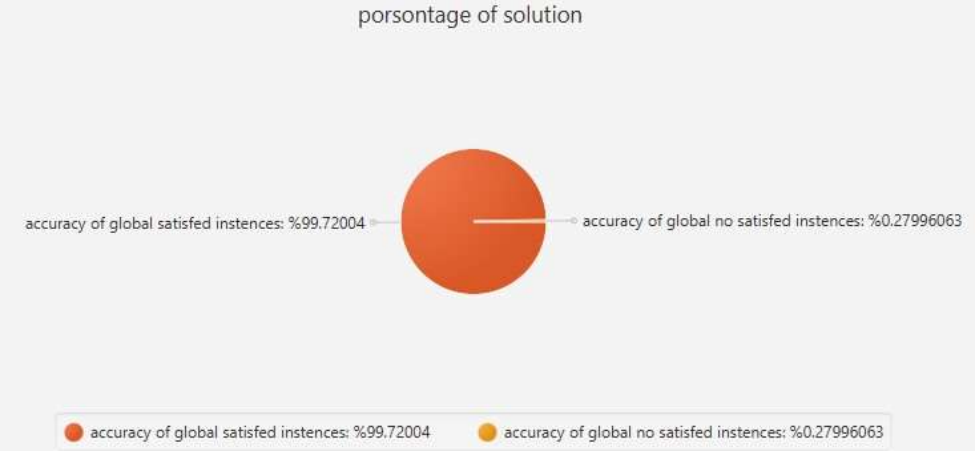
  Dans cet algorithme nous avons utilisé plusieurs paramètres, **TimeOut** pour éviter de tombé dans des cas ou l'algorithme boucle d'une manière infinie où il prendre beaucoup de temps pour trouver une solution.
  Le deuxième paramètre est le **nombre d'instances** qu'il doit résolu.
  PopulationSize définie le nombre des solutions initiale (espace de solutions).
  Crossover et **Mutation Chance** pour définir le taux de croissement et de mutation.
  SelectionSize représente le nombre des solutions qu'on doit sélectionner parmi les solutions de population.

- L'exécution de l'algorithme

  Apres l'exécution on obtient le résultat suivant.


DFS Algo
A star Algo
GA Algo
ACO Algo
Comparaision graph
POPULATION SIZE
75
CROSSOVER CHANCE
0,95
MUTATION CHANCE
0,05
SELECTION_SIZE
30
number of instaces:
100
TimeOut:
2 000
4 000
6 000
8 000
10 000
RUN
Time:2.5336 min
accuracy of solution
accuracy
instences
accuracy of GA

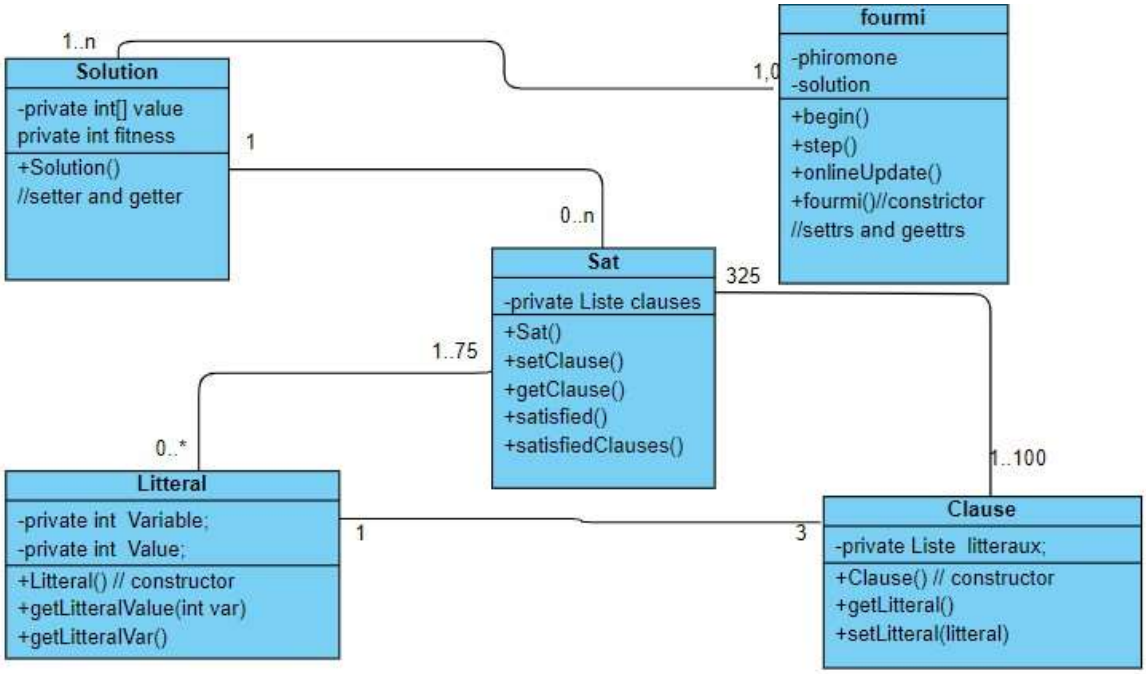

time exection for each instence
time execution
instences
Time exe for each instence in ms (GA)


porsontage of solution
accuracy of global satisfed instences: %99.72004
accuracy of global no satisfed instences: %0.27996063
accuracy of global satisfed instences: %99.72004
accuracy of global no satisfed instences: %0.27996063

(i) Ant Colony Optimization Algorithm (**ACO**)

L'optimisation par colonie de fourmis ou OCF est un algorithme récent, introduit et utilisé pour résoudre le problème d'ordonnancement dans des ateliers de type job shop. Cet algorithme est dérivé du comportement de fourmis, Il a fait preuve d'une rapidité de convergence, lui permettant d'être appliqué à plusieurs classes de problèmes d'optimisation [searchgate].

Nous avons implémenté cet algorithme pour résoudre le problème de 3-SAT. Par apport à notre modélisation précédent nous avons ajouté une petite modification, la class **Ant** (fourmi) qui lui va trouver la meilleur solution pour notre problème sat.



- Les paramètres utilisés dans l'algorithme

  Dans cet algorithme nous avons utilisé plusieurs paramètres, **TimeOut** pour éviter de tombé dans des cas ou l'algorithme boucle d'une manière infinie où il prendre beaucoup de temps pour trouver une solution.

  Le deuxième paramètre est le **nombre d'instances** qu'il doit résolu.

  Ant Count définie le nombre des fourmis initiale (espace de solutions).

  Alfa, Beta, V_rate sont des paramètres empirique qui ont un rôle sur le update de phéromone.

  Max_Iteration représente le nombre itération globale pour trouver une solution.

- L'exécution de l'algorithme

  Apres l'exécution on obtient le résultat suivant.


accuracy of global satisfed instences: %99.292366
accuracy of global no satisfed instences: %0.707634
accuracy of global satisfed instences: %99.292366
accuracy of global no satisfed instences: %0.707634
DFS Algo
A star Algo
GA Algo
ACO Algo
Comparaision graph
ALPHA
0,3
BETA
0,1
V_RATE
0,1
MAX_ITERATIONS
50000
nbr of instaces:
100
ANT_COUNT
100
Time Out:
2 000
4 000
6 000
8 000
10 000
RUN
time :3 min
accuracy of solution
accuracy
instences
accuracy of ACO
time exection for each instence
time execution
instences
Time exe for each instence in ms (ACO)

III) Conclusion

(i) Tableaux de comparaison :

Time out = 2000 ms	DFS	A star	Genetic algorithme	Ant colony optimisation
Accuracy (%)	91.41541	99.41852	99.72004	99.292366
Time execution (min)	3.33335	2.80945	2.5336	3

(ii) Conclusion générale :

A travers ce projet nous avons vu en appliquant les algorithmes sur des problèmes de taille intéressent qu'il n'est souvent pas possible de résoudre ces problèmes en utilisant des méthodes de résolution exhaustives ou purement algorithmiques car ces derniers sont couteuses en terme de temps et/ou espace mémoire ; par contre l'utilisation ; des autres méthodes plus performants comme les métas heuristique est clairement nécessaire.

Références

ALGORITHME A COLONIES DE FOURMIS POUR L'OPTIMISATION DE L'ORDONNANCEMENT DANS LES ATELIERS JOB SHOP. (s.d.). Récupéré sur www.researchgate.net.

Algorithme génétique. (s.d.). Récupéré sur fr.wikipedia.org.

https://www.hurna.io/fr/academy/algorithms/maze_pathfinder/dfs.html. (s.d.). Récupéré sur www.hurna.io.

https://www.techno-science.net/definition/6469.html. (s.d.). Récupéré sur www.techno-science.net.

TRYSTRAM, D. (2011). *Le problème SAT et ses variantes.* Grenoble : Master Informatique.