Module :Intelligence en essaim

Rapport du mini projet :

« Développement des solvers SAT »

Spécialité : Master 1 SII

Etudiante :

CHIKH Khadidja

15/07/2019

Introduction :

Occupant une place centrale dans le domaine de l'intelligence artificielle, le problème de la satisfiabilité ou SAT s'interesse à la satisfiabilité des formules logiques de manière générale et celle des formules CNF(*Conjunctive Normal Form*) d'ordre zéro en particulier. SAT peut être défini comme suit :
Instance : X un ensemble de variables booléennes, C un ensemble de clauses qui sont des disjonctions de littéraux, un littéral est une variable appartenant à X avec ou sans le connecteur de négation.
Question : Existe-il une instanciation de X telle que la conjonction des clauses de C est vraie ? **[3]**

Pour répondre à cette question, il nous faut d'abord connaître nos données (instances) ensuite chercher la solution. Pour le SAT il existe 2 types d'instances : Satisfiables et non satisfiables (dans ce projet, nous manipulons des instances satisfiables, à des solutions de taille 20,50 et 75).

Objectif : Implémenter des solutions intelligentes pour le problème de satisfiabilité, en se basant sur des méthodes heuristiques et des algorithmes évolutionnaires tels que l'algorithme génétique et le PSO (*Particle Swarm Optimization*).

Modélisation adoptée :

Nous avons adopté la structure de matrice d'entiers pour la représentation de l'instance : chaque ligne représente une clause et chaque élément de la matrice représente un littéral.

1-Les méthodes heuristiques :

Modélisation :

Espace des états :
Etats : Toutes les configurations possibles.
Etat initial : Le nœud initial (nous l'avons initialisé ayant la valeur 0).
Etat final: Toute configuration possible différente de l'état initial. La solution est obtenue en parcourant la chaine de cet état vers l'état initial.
Successeurs : Ce sont les nœuds suivants,représentant chacun le prochain état (ils sont insérés de façons symmétrique ex : 0 12 -12 ..etc).

Recherche en largeur d'abord (*Breadth First Search) :*

Il s'agit de consruire un arbre suivant la stratégie FIFO (*First In First Out)* i.e. L'arborescence est construite de manière horizontale. Nous utilisons donc une file selon cette stratégie pour stocker les nœuds (états) à développer (qui n'ont pas encore été explorés) .**[1]** Nous défilons le premier nœud et determinons ses deux fils ensuite les enfilons, en gardant leurs chainages qui mènent vers le nœud parent. Nous répétons ce processus tant que la file n'est pas vide et qu'il n'existe pas d'état final parmi ces fils. La solution est la chaine allant du nœud courant vers le premier parent(racine).

Recherche en profondeur d'abord (*Depth First Search) :*

Il s'agit de consruire un arbre suivant la stratégie LIFO (*Last In First Out)* i.e. L'arborescence est construite de manière verticale. Lorsqu'un certain seuil de profondeur est atteint, le processus considère un nœud du niveau précédent. Nous utilisons donc une pile pour les nœuds à développer. **[2]** Nous dépilons le premier nœud et nous vérifions si nous avons atteint le seuil de profondeur (alors nous dépilons encore..). Nous déterminons ses deux fils ensuite les empilons en gardant leurs chainages qui mènent vers le nœud parent.Nous répétons ce processus tant que la file n'est pas vide et qu'il n'existe pas d'état final parmi ces fils. La solution est la chaine allant du nœud courant vers le premier parent(racine).

Expérimentations :

Durant les experimentations, nous avons remarqué que l'algorithme de recherche en largeur ainsi que l'algorithme de recherche en profondeur prennent un temps important pour trouver la solution optimale. Nous avons donc rajouté une contrainte de temps T (l'algorithme s'arrête peu importe l'état obtenu tant qu'il n'est pas un état final). Les résultats obtenus (meilleurs) étaient comme suit :

Recherche en largeur d'abord :

Sur instance à taille de solution=20: T=30s, 85/91 clauses satisfaites.

Sur instance à taille de solution=50: T=120s, 114/218 clauses satisfaites.

Recherche en profondeur d'abord :

Sur instance à taille de solution=20: T=30s, 85/91 clauses satisfaites.

Sur instance à taille de solution=50: T=120s, 196/218 clauses satisfaites.

2-Algorithme génétique pour SAT :

Modélisation :

Solution :

La solution est représentée par une entité ayant comme caractéristiques : Un tableau binaire appelé « bits » représentant une solution à l'instance et un attribut appelé *fitness* représentant le nombre de clauses satisfaites par « bits ».

Espace des solutions :

Ou population de solution, est un ensemble de solutions.

Taille et complexité :

Taille : C'est la taille de la population de solution.

Complexité : nombre_iterations*(taillepopulation*(cr+mr) +taillepopulation)=O(nombreitérations)

Algorithme génétique :

Les algorithmes génétiques (AGs) sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle et de la génétique. Leur fonctionnement est extrêmement simple. Nous partons avec une population de solutions potentielles (chromosomes) initiales arbitrairement choisies. Nous évaluons leur performance (fitness) relative. Sur la base de ces performance nous créons une nouvelle population de solutions potentielles en utilisant des opérateurs évolutionnaires simples : le croisement, la mutation et la sélection. Nous recommençons ce cycle jusqu'à ce que nous trouvons une solution satisfaisante. **[4]**

Croisement pour SAT :

Nous engendrons aléatoirement deux solutions de la population et nous appliquons le croisement, en rajoutant les deux solutions obtenues à la population, et cela pour un nombre de fois égal au taux de croisementn dans une même itération.

Mutation pour SAT :

Nous engendrons aléatoirement une solution(gène) et nous modifions une valeur(chromosome) de son « bits » et nous rajoutons la nouvelle solution à la population, et cela pour un nombre de fois égal au taux de mutation dans une même itération.

Selection pour SAT :

En éliminant par exemple les solutions les moins bonnes à la fin de l'itération.

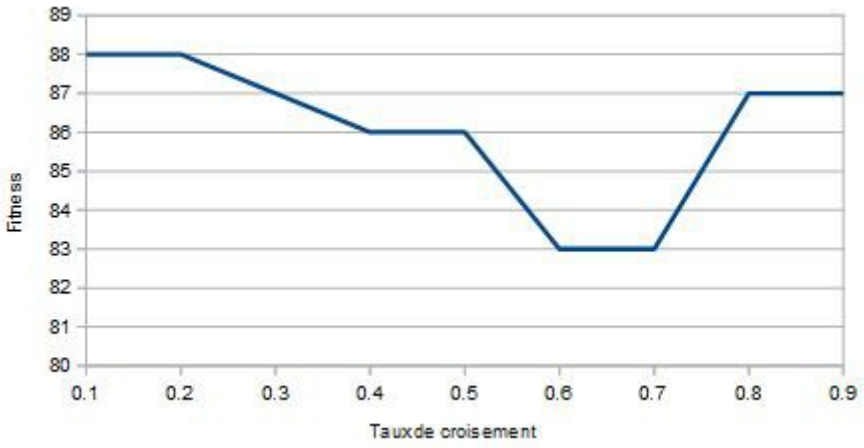
Experimentations :

Réglages des paramètres:

Sur instance à taille de solution=20:

Recherche du taux de croisement (cross rate) :

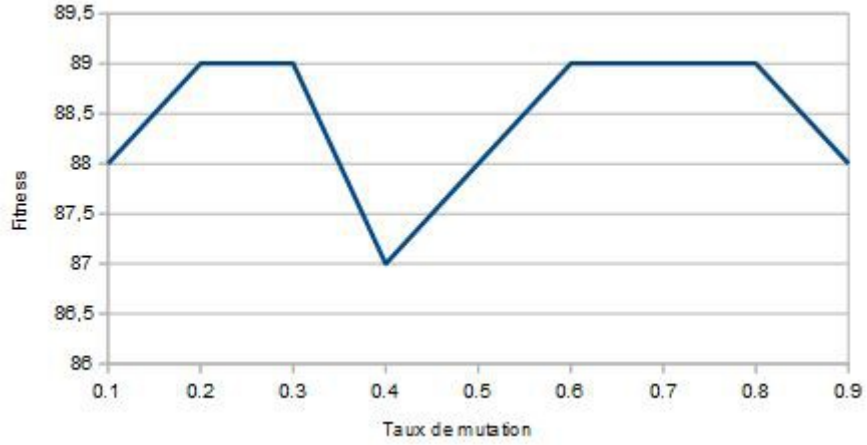
En commençant par la combinaison: nombre d'itérations=10, taille de population=10, taux de mutation=0,1 (10%) , taux de croisement=0,1 (10%) et en augmentant ce dernier par pas de 0,1 jusqu'à 0.9, nous avons obtenu les résultats suivants :



C'est clair que pour cr=0,1 et 0,2 nous avons obtenu la meilleure *fitness* (88), et en raison d'autres observations (réexecutions) qui ont montré une favorisation entre les deux, la valeur prise est **cr=0,1.**

Recherche du taux de mutation (*mutation rate*) :

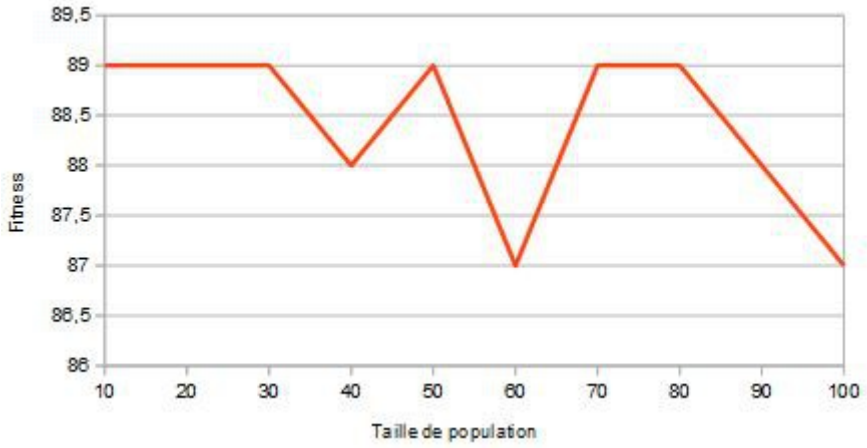
En commençant par la combinaison: nombre d'itérations=10, taille de population=10, taux de croisement=0,1 (10%) , taux de mutation=0,1 (10%) et en augmentant ce dernier par pas de 0,1 jusqu'à 0.9, nous avons obtenu les résultats suivants :



C'est clair que pour mr=0.2, 0.3, 0.6 -0.8 nous avons obtenu la meilleure *fitness*(89), et en raison d'autres observations (réexecutions) qui ont montré une favorisation entre ces valeurs, la valeur prise est **mr=0,3.**

Recherche de la taille de population *(population size)* :

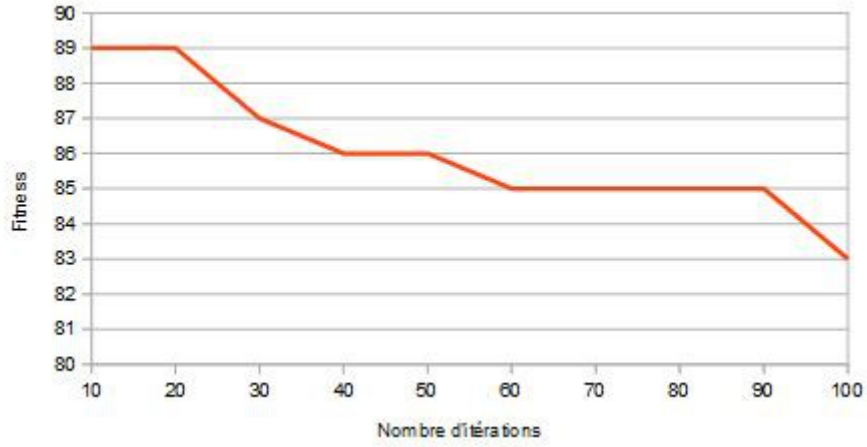
En commençant par la combinaison: nombre d'itérations=10, taux de croisement=0,1 (10%) , taux de mutation=0,3 (30%), taille de population=10 et en augmentant ce dernier par pas de 10 jusqu'à 100, nous avons obtenu les résultats suivants :



C'est clair que pour popsize=10-30, 50, 70,80 nous avons obtenu la meilleure *fitness*(89), et en raison d'autres observations (réexecutions) qui ont montré une favorisation entre ces valeurs, la valeur prise est popsize**=50.**

Recherche du nombre maximum d'itérations :

En commençant par la combinaison: taux de croisement=0,1 (10%) , taux de mutation=0,3 (30%), taille de population=50, nombre d'itérations=10 et en augmentant ce dernier par pas de 10 jusqu'à 100, nous avons obtenu les résultats suivants :



C'est clair que pour itérations=10 et 20 nous avons obtenu la meilleure *fitness*(89), et en raison d'autres observations (réexecutions) qui ont montré une favorisation entre ces valeurs, la valeur prise est maxiter**=10.**

Remarque : Par manque de temps, nous n'avons pas pu réaliser des expérimentations sur les instances à taille 50 ou 75. Ceci dit les paramètres auront d'autres valeurs.

Performances :

Qualité de la solution : La meilleure solution obtenue satisfait 89 clauses (fitness maximale) ce qui n'est pas optimal.

Temps d'execution : Nous avons remarqué que l'algorithme génétique prend un temps important relatif à l'augmentation des valeurs des paramètres et plus précisément le paramètre **maxiter.**

Au cours des experimentations, nous avons remarqué qu'à un certain moment les résultats stagnent. Et c'est pour ça,qu'il faut à chaque étape de l'algorithme, effectuer le compromis entre **explorer** l'espace de recherche, afin d'éviter de stagner dans un optimum local, et **exploiter** les meilleurs individus obtenus, afin d'atteindre de meilleurs valeurs aux alentours. Trop d'exploitation entraîne une convergence vers un optimum local, alors que trop d'exploration entraîne la non-

convergence de l'algorithme. Typiquement, les opérations de sélection et de croisement sont des étapes d'exploitation, alors que l'initialisation et la mutation sont des étapes d'exploitation. On peut ainsi régler les parts respectives d'exploration et d'exploitation en jouant sur les divers paramètres de l'algorithme.**[5]**

3-Algorithme PSO(*Particle Swarm Optimization*) pour SAT :

Modélisation :

Position : Un tableau binaire, représentant une solution à l'instance.
Vitesse : Un nombre entier.

Algorithme PSO :

Chaque solution de l'espace de recherche stimule une particule, a une position, une vitesse et agit selon la meilleure position (but). Nous engendrons aléatoirement un groupe de particules, évaluons leurs positions, calculons la meilleure position(la plus satisfaisante) en utilisant la loi de transition, ensuite pour chaque itération nous modifions les vitesses et les positions, evaluons leurs fitness et nous modifions à la fin la meilleure position. Nous répétons cela jusqu'à trouver la solution optimale.

Position pour SAT :

Représente la solution atteinte par une particule à un temps donné.

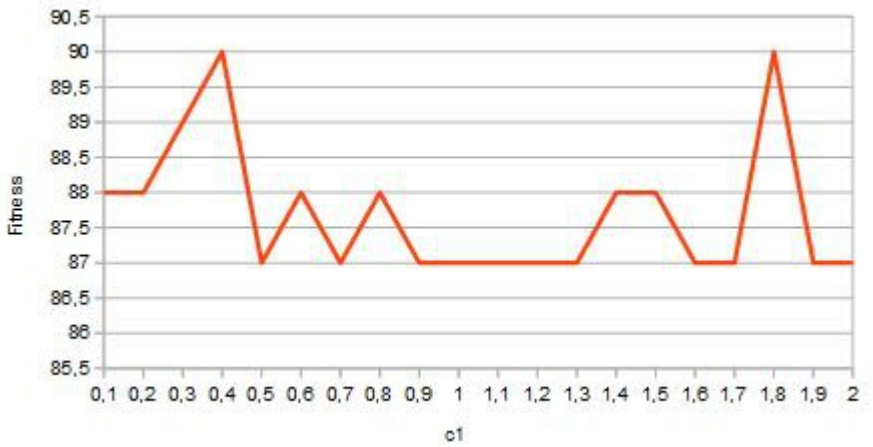
Vitesse pour SAT :

Le nombre selon lequel nous allons modifier la position, caractérisée par des constantes w, c1,c2,r1,r2.

Expérimentations :

Recherche de c1 :

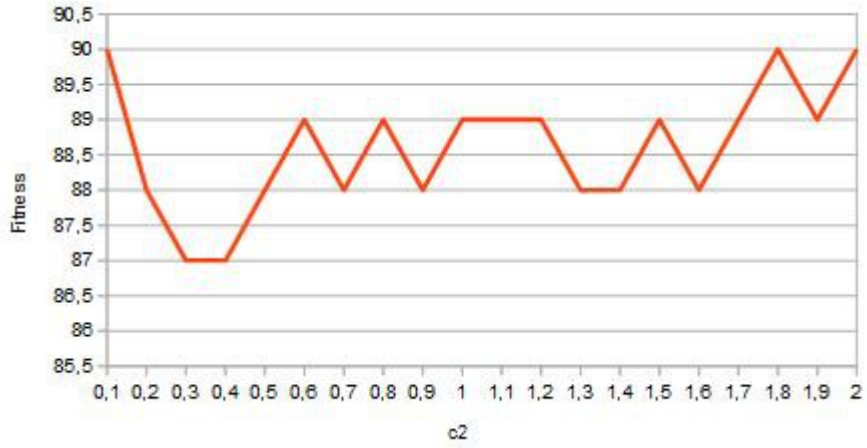
En commençant par la combinaison: nombre d'itérations=10, taille de swarm=10, r1=0,1, r2=0,1, w=0,1, c2=0,1, c1=0,1 et en augmentant ce dernier par pas de 0,1 jusqu'à 2, nous avons obtenu les résultats suivants :



C'est clair que pour c1=0,4 et 1,8 nous avons obtenu la meilleure *fitness*(90), et en raison d'autres observations (réexecutions) qui ont montré une favorisation entre ces valeurs, la valeur prise est **c1=0,4.**

Recherche de c2 :

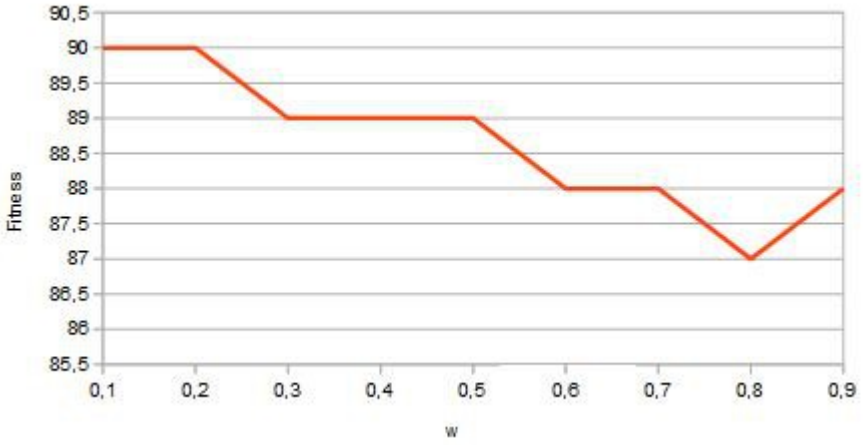
En commençant par la combinaison: nombre d'itérations=10, taille de swarm=10, r1=0,1, r2=0,1, w=0,1, c1=0,4, c2=0,1 et en augmentant ce dernier par pas de 0,1 jusqu'à 2, nous avons obtenu les résultats suivants :



C'est clair que pour c2=1,8 et 2 nous avons obtenu la meilleure *fitness*(90), et en raison d'autres observations (réexecutions) qui ont montré une favorisation entre ces valeurs, la valeur prise est **c2=2.**

Recherche de w :

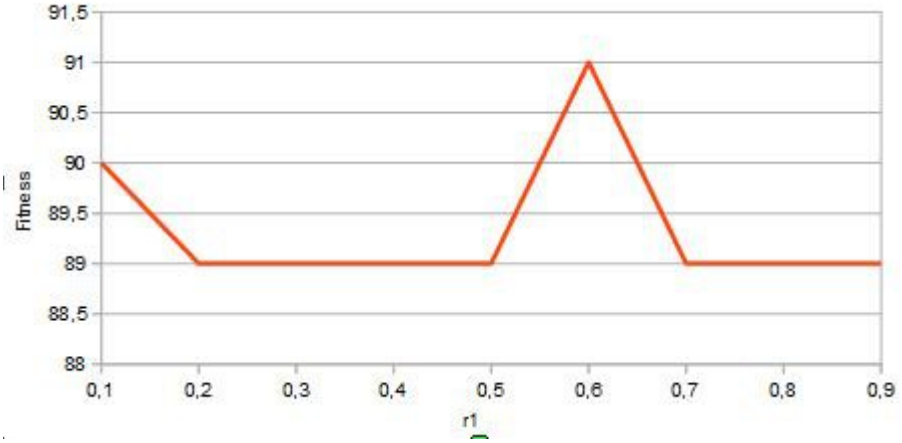
En commençant par la combinaison: nombre d'itérations=10, taille de swarm=10, r1=0,1, r2=0,1, c1=0,4, c2=2 , w=0,1 et en augmentant ce dernier par pas de 0,1 jusqu'à 0,9, nous avons obtenu les résultats suivants :



C'est clair que pour w=0,1 et 0,2 nous avons obtenu la meilleure *fitness*(90), et en raison d'autres observations (réexecutions) qui ont montré une favorisation entre ces valeurs, la valeur prise est **w=0,2.**

Recherche de r1:

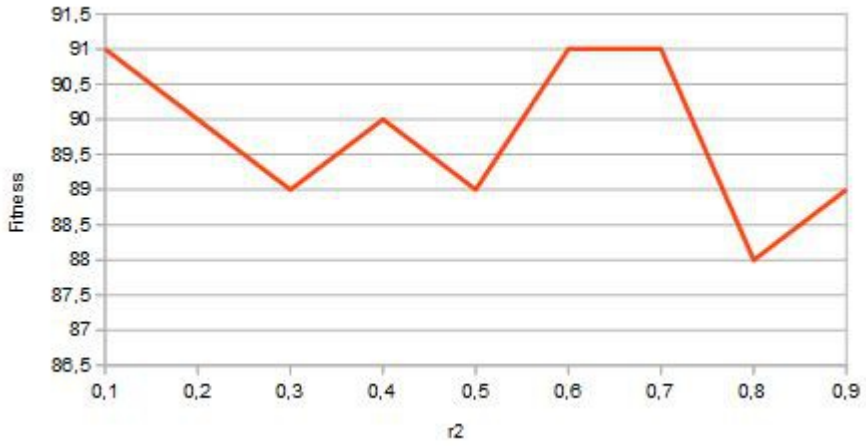
En commençant par la combinaison: nombre d'itérations=10, taille de swarm=10, r2=0,1, c1=0,4, c2=2 , w=0,2, r1=0,1 et en augmentant ce dernier par pas de 0,1 jusqu'à 0,9, nous avons obtenu les résultats suivants :



C'est clair que pour r1=0,6 nous avons obtenu la meilleure *fitness*(91) et l'optimale. La valeur prise est **r1=0,6.**

Recherche de r2:

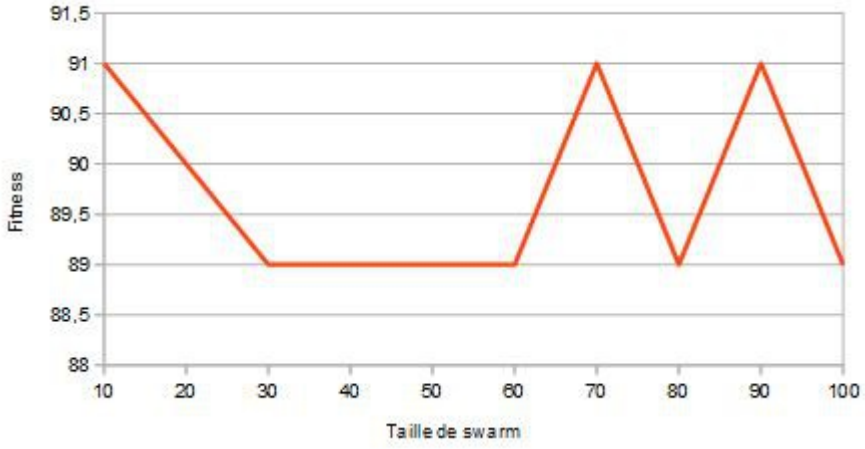
En commençant par la combinaison: nombre d'itérations=10, taille de swarm=10, c1=0,4, c2=2 , w=0,2, r1=0,6, r2=0,1 et en augmentant ce dernier par pas de 0,1 jusqu'à 0,9, nous avons obtenu les résultats suivants :



C'est clair que pour r2=0,1, 0,6 et 0,7 nous avons obtenu la meilleure *fitness*(91) et l'optimale , et en raison d'autres observations (réexecutions) qui ont montré une favorisation entre ces valeurs, la valeur prise est **r2=0,6.**

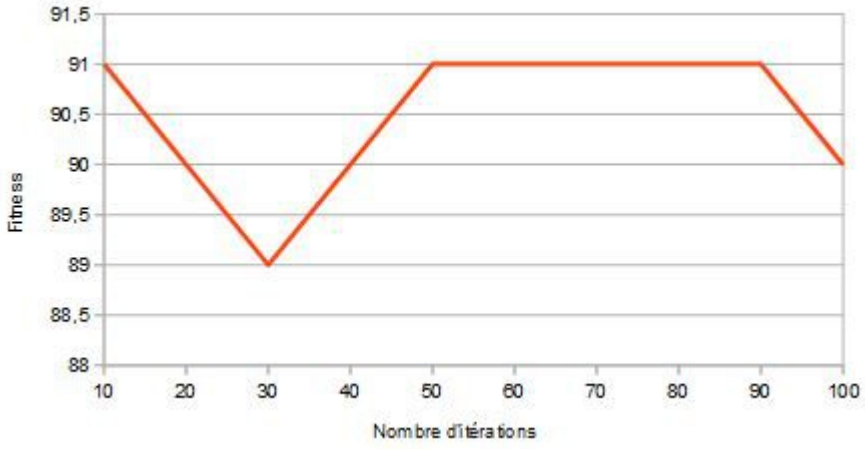
Recherche de la taille du swarm*(swarm size):*

En commençant par la combinaison: nombre d'itérations=10, c1=0,4, c2=2 , w=0,2, r1=0,6, r2=0,6, taille de swarm=10 et en augmentant ce dernier par pas de 10 jusqu'à 100, nous avons obtenu les résultats suivants :



C'est clair que pour swarmsize=10, 70 et 90 nous avons obtenu la meilleure *fitness*(91) et l'optimale , et en raison d'autres observations (réexecutions) qui ont montré une favorisation entre ces valeurs, la valeur prise est **swarmsize=70.**

Recherche du nombre maximum d'itérations*:*
En commençant par la combinaison: c1=0,4, c2=2 , w=0,2, r1=0,6, r2=0,6, taille de swarm=70, nombre d'itérations=10 et en augmentant ce dernier par pas de 10 jusqu'à 100, nous avons obtenu les résultats suivants :



C'est clair que pour maxiter=10, 50-90 nous avons obtenu la meilleure *fitness*(91) et l'optimale , et en raison d'autres observations (réexecutions) qui ont montré une favorisation entre ces valeurs, la valeur prise est **maxiter=70.**

Remarque : Par manque de temps, nous n'avons pas pu réaliser des expérimentations sur les instances à taille 50 ou 75. Ceci dit les paramètres auront d'autres valeurs.

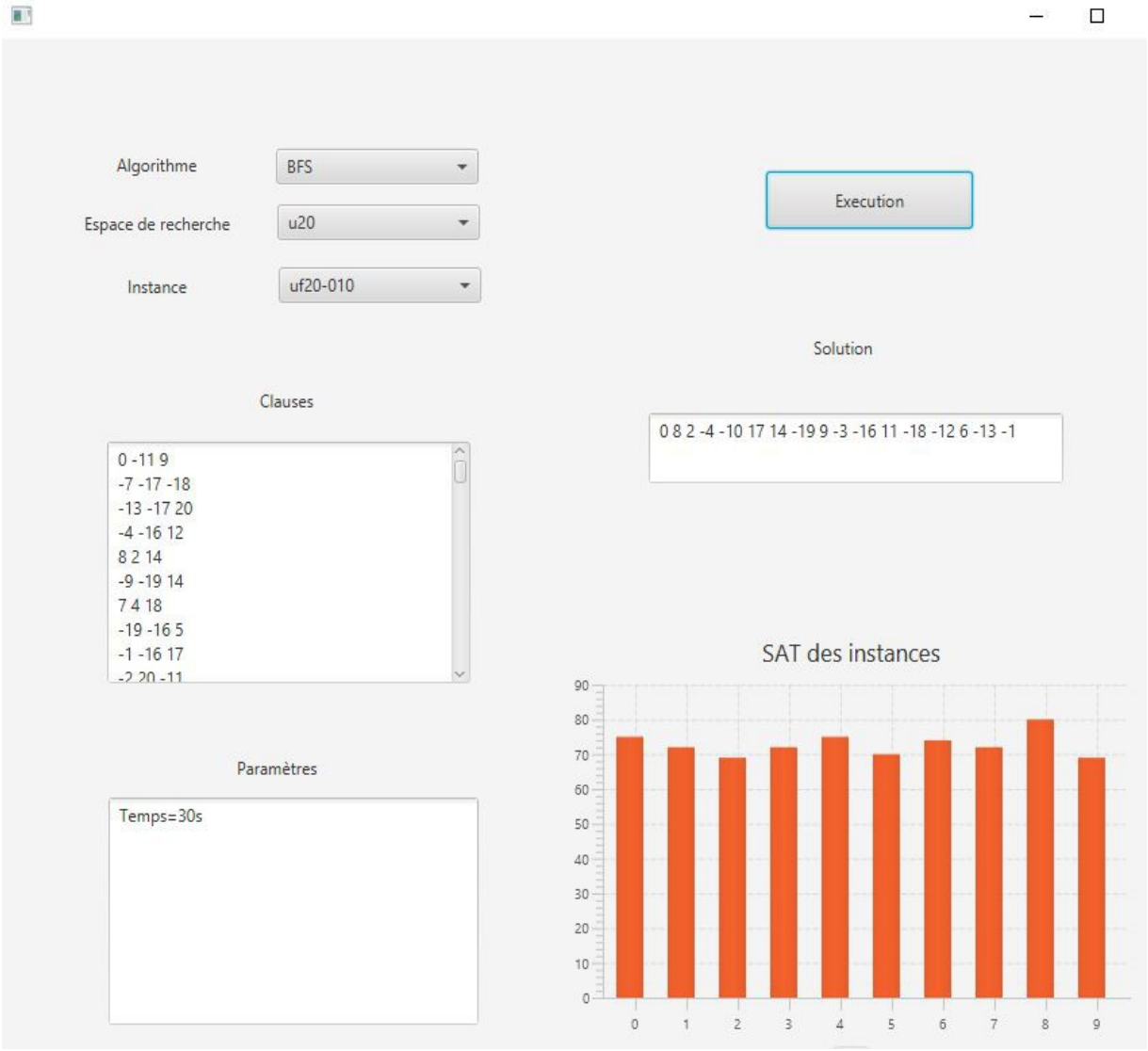
3-Interface graphique :

L'interface permet de selectionner un algorithme, un espace de recherche (u20,u50,u75) et une instance à la fois. L'execution affiche les clauses de l'instance, la solution, les paramètres de l'algorithme et le graphe de satisfiabilité, en fonction des instances(x) et les nombre de clauses satisfaites à chaque fois. Ce graphe est construit au fur et à mesure, c'est-à-dire que pour un algorithme choisi, pour chaque execution avec une autre instance nous aurons la barre de celle-ci ajouté au graphe.

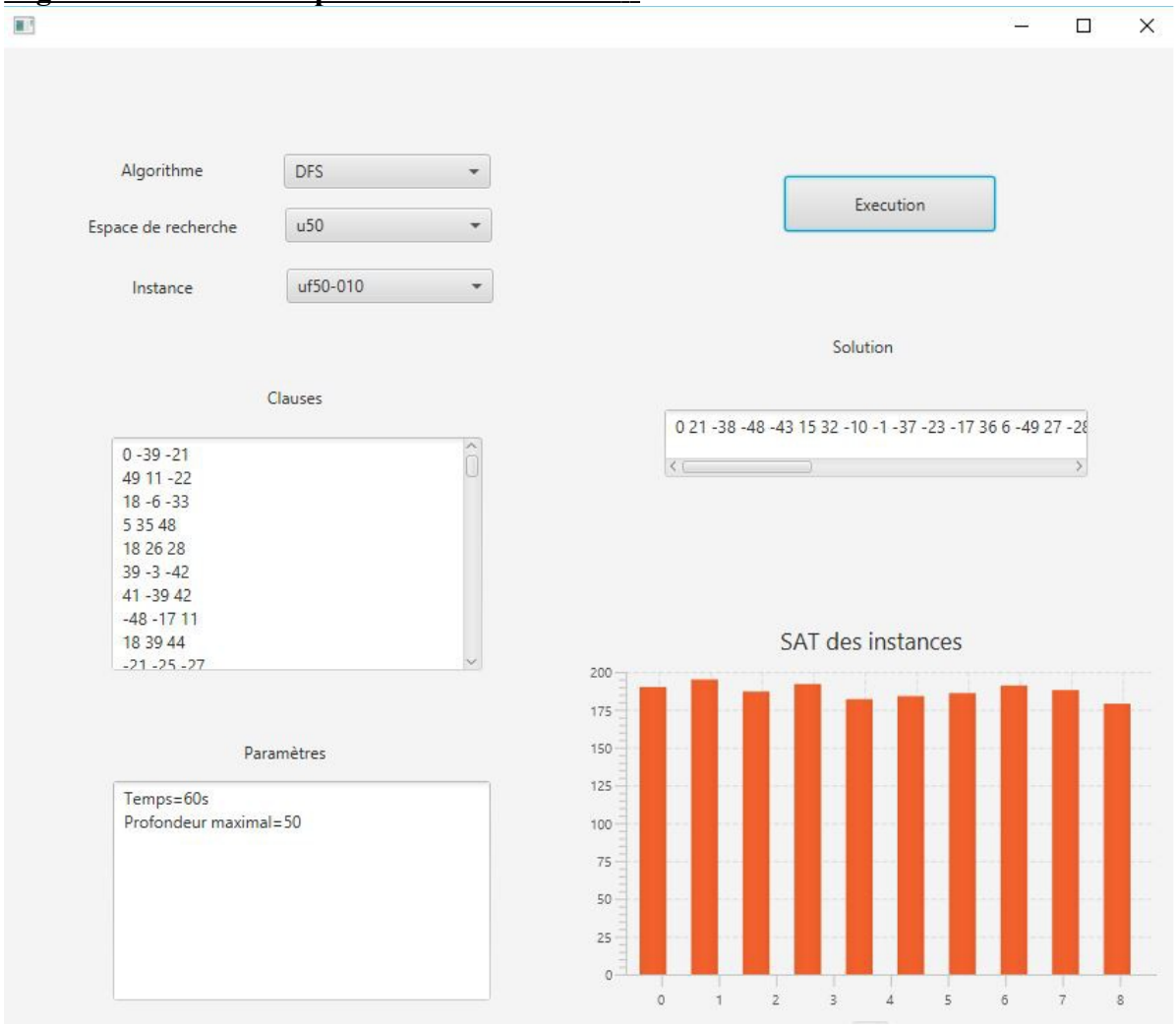
Executions :

Pour BFS et DFS, nous avons fixé le temps à 30s pour u20 et 60s pour u50.

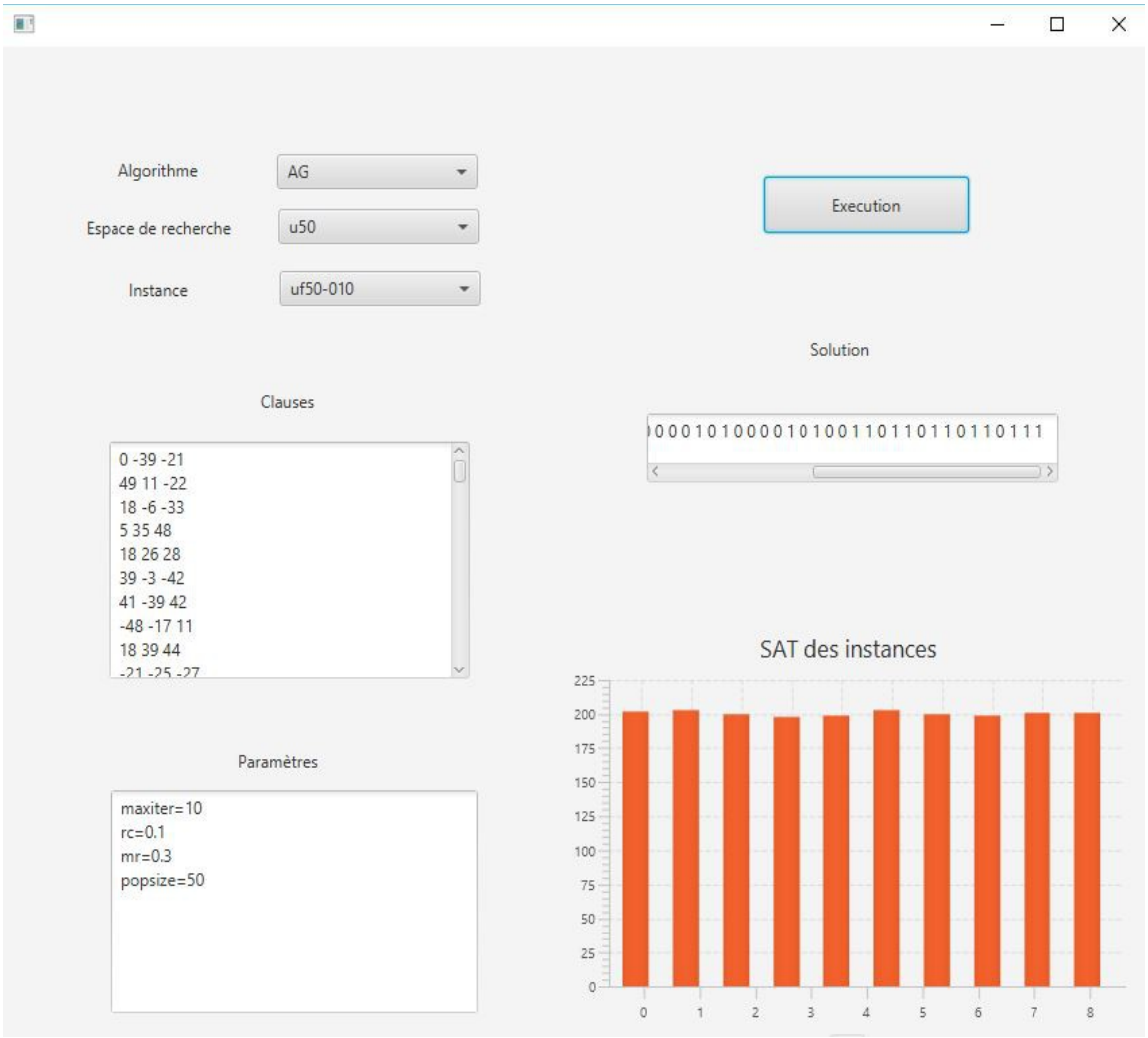
Algorithme BFS avec epace de recherche=u20 :



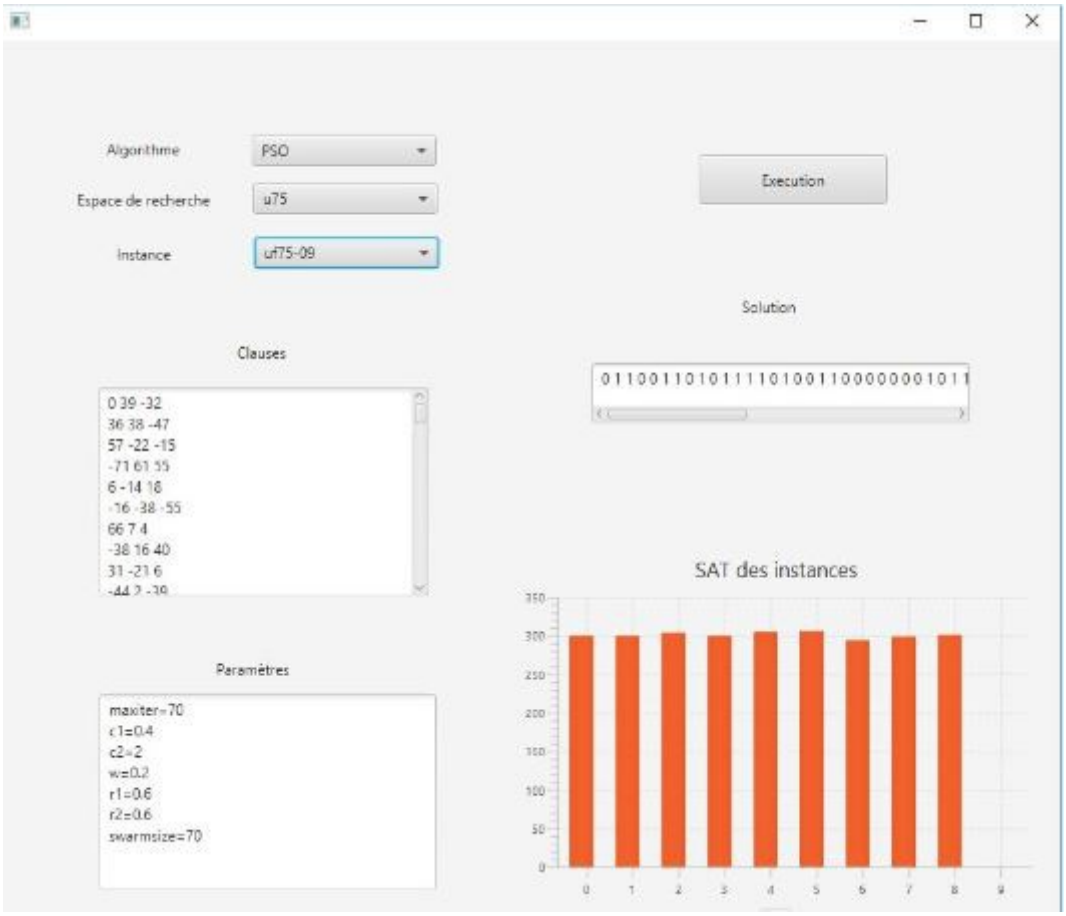
Algorithme DFS avec epace de recherche=u50 :



Algorithme génétique avec epace de recherche=u50 :



Algorithme PSO avec epace de recherche=u75 :



Conclusion :

D'après les expérimentations faites sur les différents algorithmes abordés dans ce projet, nous pouvons déduire certaines conclusions :

-Les BFS et DFS ont nécessité une limitation de temps, et ont engendré des solutions à 85 clauses satifaites au maximum.

-L'algorithme génétique prend beaucoup plus de temps et ceci est dû a sa complexité qui peut grandir d'une façon importante résultante des différentes opérations impliquées et les structures necessaires, engendrant une solution à 89 clauses satisfaites au maximum.

-L'algorithme PSO était le plus rapide et le seul qui a atteint une solution optimale.

REFERENCES :

[1] *Algorithmique Moderne, Analyse et Complexité , H.ZERKAOUI DRIAS*- « Problèmes combinatoires et méthodes heuristiques - Recherche en largeur d'abord»

[2] *Algorithmique Moderne, Analyse et Complexité , H.ZERKAOUI DRIAS*- « Problèmes combinatoires et méthodes heuristiques - Recherche en profondeur d'abord»

[3] *Algorithmique Moderne, Analyse et Complexité , H.ZERKAOUI DRIAS*- « Introduction aux problèmes NP-complets - Le problème de la satisfiabilité ou SAT»

[4] Document : **«** Présentation des algorithmes génétiques et de leurs applications en économie » , [http://deptinfo.unice.fr/twiki/pub/Minfo04/IaDecision0405/Prsentationdesalgorithmesgntiquesetdeleursapplicationsenconomie_P.pdf]

[5] « Algorithmes évolutionnaires et problèmes inverses » [http://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/poly/cours009.html]