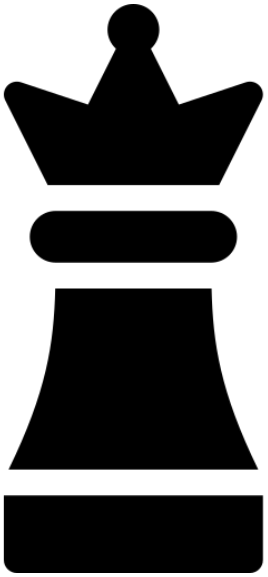
28 janvier 2022



TP SAT N-Queens

[Algorithmes de recherches]

Lien du projet : https://github.com/aperrier004/SAT_N-Queens

PERRIER ALBAN

BORDEAUX INP

ENSC 3A – IA

Introduction

Pendant les séances de cours, nous avons travaillé sur le jeu du sudoku, j'ai ainsi choisi de réaliser le TP d'algorithmes de recherche noté sur le problème des N-reines. J'ai ainsi pu modéliser ce problème en SAT, en raisonnant par contraintes.

Le travail réalisé

J'ai repris le setup mis en place en cours, avec notamment la librairie pysat fournie. J'ai évidemment commencé par solutionner le problème des 8-reines, en codant la fonction de génération de contraintes, pour énumérer toutes celles du problème.

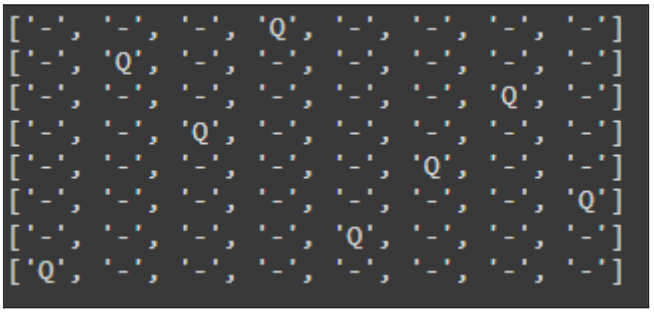


Figure 1 - Solution au problème des 8-reines

Une fois que j'ai réussi à afficher les solutions pour les 8-reines, j'ai cherché à comparer les résultats en augmentant le nombre de reines sur le chessboard.

Ainsi, j'ai séparé les responsabilités dans le code en créant des fonctions pour afficher le chessboard, et une autre résoudre la solution.
Avec cela, j'ai pu simplement avec une boucle *for* utiliser le solveur sur un nombre différent de reine.

J'ai alors essayé d'interpréter les données que la librairie me fournissait, notamment sur le running time, j'ai alors réalisé ce graphique :

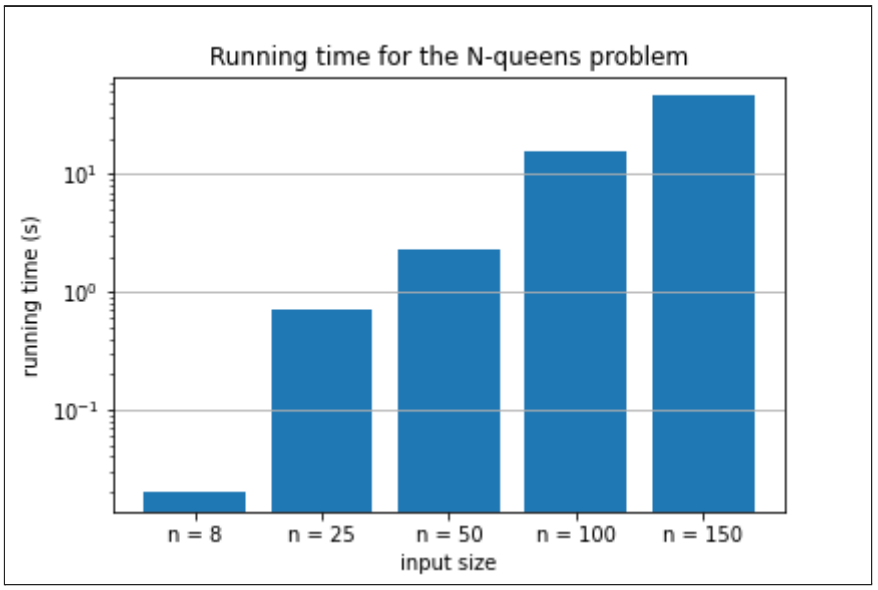


Figure 2 - Comparaison du running time par rapport au nombre de reines

Et j'ai ensuite comparé le nombre de variables et de clauses avec celui-ci :

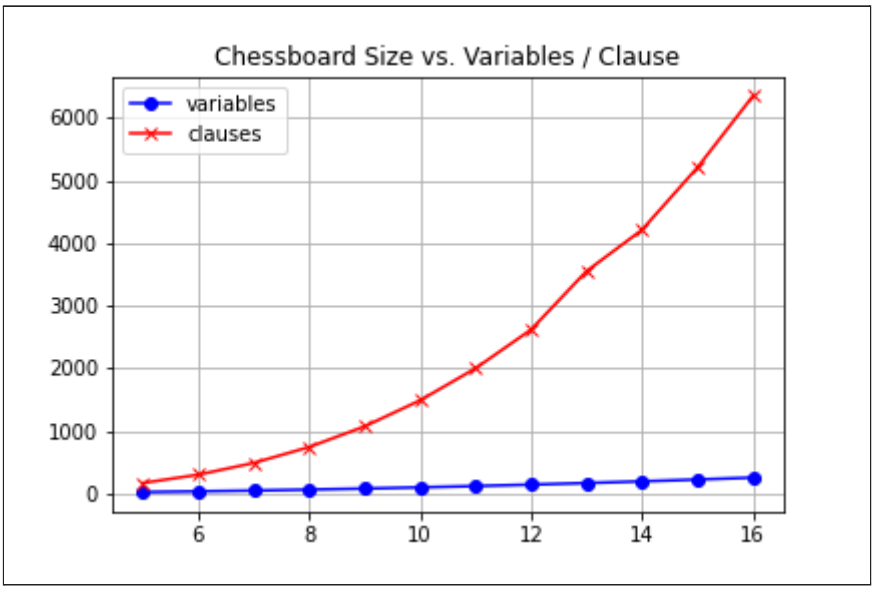


Figure 3 - Comparaison du nombre de variables et de clauses par rapport au nombre de reines

Cette petite étude comparative m'a permis de mieux comprendre les enjeux d'un SAT, et de voir à partir de quand l'utilisation de celui-ci n'était plus rentable pour solutionner le problème des N-reines.

J'ai également essayé d'aller plus loin dans ce TP en ajoutant de nouvelles règles. Je souhaitais ajouter la notion de "superqueens", où l'objectif pour le placement des reines est de couvrir tous les déplacements d'un éventuel cavalier sur le chessboard, en plus des contraintes normales.
Je n'ai cependant pas réussi cette dernière partie, n'ayant pas réussi à modéliser les contraintes de déplacement des cavaliers.
Je voulais essayer également une approche plus simple où je donnerai simplement la position de cavaliers fixement sur le chessboard, mais je ne savais pas comment mettre cette contrainte sur un chessboard N x N (plutôt que juste 8x8 plus classiquement).

Dans le cas d'un chessboard 8x8 la position des cavaliers à couvrir initialement pourraient ainsi être celle-ci :

	A	B	C	D	E	F	G	H
1		C1					C2	
2				C1	C2			
3	C1		C1			C2		C2
4								
5								
6	C3		C3			C4		C4
7				C3	C4			
8		C3					C4	

Figure 4 - Position des cavaliers à couvrir

Après en avoir rediscuté avec mon professeur, j'ai compris que je n'avais pas correctement réfléchi au problème, j'ai donc suivi sa proposition de plutôt compter le nombre de solutions possible pour le problème des n-reines.

Ainsi, j'ai créé de nouvelles fonctions me permettant de rajouter la dernière solution trouvée par le solver en tant que contrainte pour le forcer à en trouver une nouvelle. J'ai donc pu voir que cela fonctionnait bien (en modifiant une partie du fichier pysat.py) et qu'il existe 92 solutions au problème des 8-reines.