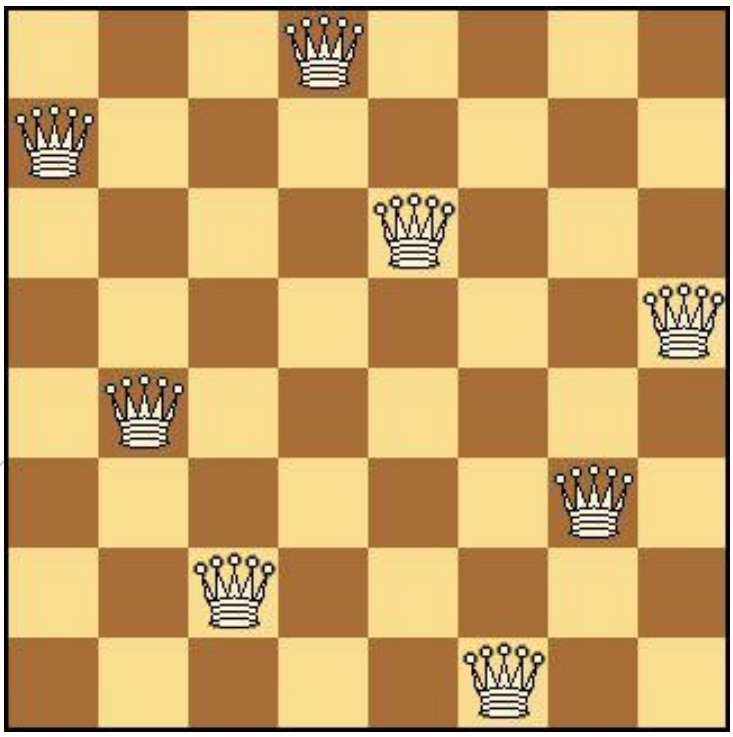
UNIVERSITÉ DE FRANCHE-COMTÉ

PROJET OC M2

Année 2016/2017

Optimisation et Complexité

« Problème des reines en SAT »



Enseignant : Alain Giorgetti

Avec la participitation de :

Mehdi Azizi, Elodie Bernard, Mathieu Briland, Anthony Casagrande, Benoit Crivelli, Wentian Huang, Vianney Lotoy Bendenge, Yassin Ousleveh Bileh, Simon Pallais, Cédric Petetin, Alexis Plumet, Pierre Wargnier

Sommaire

Introduction

Le but de ce projet est d'expérimenter les techniques SAT sur le problème des n reines en utilisant un solveur SAT.
Le problème des n-reines est un problème inspiré des échecs, proposé initialement en 1848 par un joueur d'échec du nom de Max Bassel. Ce problème consiste à disposer n reines sur un échiquier n× n de manière à ce qu'aucune d'entre elles ne soit en prise avec les autres.
Il nous faut donc effectuer des tests pour voir la faisabilité ou non des possibilités en fonction de ce nombre « n ».
Il nous faut tout d'abord définir les contraintes du problème et ensuite affecter des valeurs aux variables de telle sorte que ces contraintes soient satisfaites.
Dans un premier temps, nous formaliserons les contraintes par des formules propositionnelles et nous verrons comment les rendre interprétables par notre solveur. Puis nous expérimenterons plusieurs solutions via la création de tests pour déterminer les solutions adéquates au problème.

Formalisation

1. Mise en forme CNF

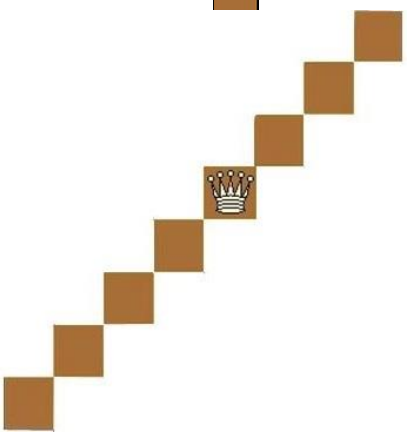
Dans un premier temps, il nous a fallu formaliser le problème, c'est-à-dire représenter le modèle sous une forme qui sera interprétable par le solveur. Pour ce faire, notre professeur nous avait mis à disposition un document comprenant les principales contraintes du problème, qui sont :

1. Il y a deux reines sur la même ligne
2. Il y a deux reines sur la même colonne
3. Il y a deux reines sur la même diagonale montante
4. Il y a deux reines sur la même diagonale descendante

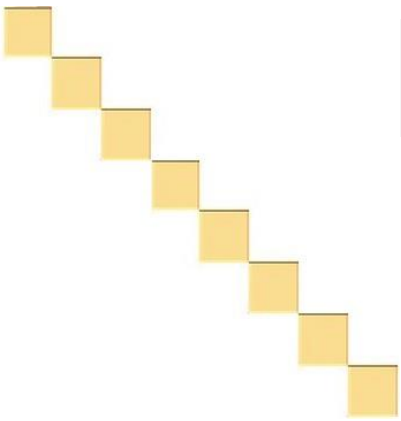
Ce qui donne, une fois les contraintes exprimées en logique propositionnelle :

$$\bigvee_{1\le i,j,k\le n,\ j<k} (r_{i,j} \wedge r_{i,k})$$

$$\bigvee_{1\le i,j,k\le n,\ i<k} (r_{k,j} \wedge r_{i,j})$$



$$\bigvee_{1\le i,j\le n-1,\ 1\le k\le n-i,\ 1\le k\le n-j} (r_{i,j} \wedge r_{i+k,j+k})$$



$$\bigvee_{1\le i,j\le n-1,\ 1\le k\le n-i,\ 1\le k\le j-2} (r_{i,j} \wedge r_{i+k,j-k})$$

Nous avons donc effectué la négation de ces formules pour obtenir les contraintes du problème. Ces contraintes traduisent donc le fait qu'il ne peut y avoir qu'une dame au maximum par ligne, colonne ou diagonale.
Ces 4 contraintes ne sont pas suffisantes pour modéliser le problème car celles-ci n'indiquent pas qu'il y'a un nombre précis de dames à placer sur le damier. Nous avons formalisé cette contrainte en partant du principe simple que s'il y'a n reines à placer sur le damier comportant n lignes et que chaque ligne ne peut contenir qu'une seule dame au maximum, alors il y a donc exactement une dame par ligne. Sachant que nous avions déjà formalisé le fait qu'il y ait un nombre de dames par ligne inférieur ou égal à 1, il nous a donc suffi de rajouter la contrainte exprimant qu'il y'a au moins une dame par ligne. Cela nous donne la formule suivante :

$$\bigwedge_{1\le i\le n}\left(\bigvee_{1\le j\le n}(r_{i,j})\right)$$

En effectuant la négation des 4 premières contraintes, on obtient directement une forme CNF du problème, et la dernière contrainte est déjà en CNF. On obtient donc au final une formalisation du problème comme suit :

$$\bigwedge_{1\le i,j,k\le n,\ j\neq k}(\neg r_{i,j}\vee\neg r_{i,k})\ \bigwedge_{1\le i,j,k\le n,\ j\neq k}(\neg r_{i,j}\vee\neg r_{k,j})\ \bigwedge_{1\le i,j\le n-1,\ 1\le k\le n-i,\ 1\le k\le n-j}(\neg r_{i,j}\vee\neg r_{i+k,j+k})\ \bigwedge_{1\le i,j\le n-1,\ 1\le k\le n-i,\ 1\le k\le j-2}(\neg r_{i,j}\vee\neg r_{i+k,j-k})\ \bigwedge_{1\le i\le n}\left(\bigvee_{1\le j\le n}(r_{i,j})\right)$$

2. Lecture par le solveur MiniSAT

Le solveur que nous utilisons, MiniSAT, a besoin que la formule soit en CNF et que le format du fichier soit sous le format DIMACS. On écrit les contraintes de notre problème dans le fichier DIMACS pour qu'elles soient respectées par le solveurs. Dans DIMACS, le formalisme veut que les ET et les OU Logiques soient représentés respectivement par 0 et un espace. De plus le signe « - » signifie la négation.

Exemple pour un damier de taille 4 :

On considère que notre damier est numéroté de 1 à 16, la première case étant celle en haut à gauche du damier, et la 16ème celle en bas à droite.

Ici, 1 représente la présence d'une reine sur la première case et -1 représente son absence.

Voici l'expression de la cinquième contrainte, elle modélise la présence d'au moins une reine par ligne :

```
1 2 3 4 0
5 6 7 8 0
9 10 11 12 0
13 14 15 16 0
```

Pour la première ligne, il ne faut pas qu'il y ait une reine sur deux cases différentes de la ligne, il en est de même pour toutes les lignes.
Voici l'expression de la première contrainte, elle modélise la présence d'au maximum une reine par ligne :

```
-1 -2 0
-1 -3 0
-1 -4 0
...
...
...
-14 -15 0
-14 -16 0
-15 -16 0
```

On retrouve le même schéma pour les lignes et colonnes suivantes.

Pour les colonnes :

```
-1 -5 0
-1 -9 0
-1 -13 0
...
...
...
-8 -12 0
-8 -16 0
-12 -16 0
```

Pour exprimer qu'il y ait au maximum une reine par diagonale, nous avons fait le choix de différencier les grandes diagonales, qui sont celles qui relient les deux cases les plus éloignées du damier, et les petites, qui sont les autres.

Grande diagonale

```
-1 -6 0
-1 -11 0
-1 -16 0
-6 -11 0
-6 -16 0
-11 -16 0
```

Les petites

```
-2 -7 0
-2 -12 0
-7 -12 0

-5 -10 0
-5 -15 0
-10 -15 0
-3 -8 0
-9 -14 0
```

Pour vérifier la présence d'une seule reine sur la diagonale, on teste toutes les combinaisons possibles deux à deux sur la diagonale.

Pour créer automatiquement ce programme en fonction de n, nous avons développé un programme en Java qui prend en paramètre la taille de l'échiquier (8 pour 8*8, 16 pour 16*16) et retourne un fichier au format DIMACS contenant toutes les contraintes nécessaires à l'interprétation par le solveur.

Nous avons donc dû générer les formules propositionnelles précédentes grâce à un programme Java, nous avons donc :

Pour les lignes (le code pour les colonnes est très similaire) :

```
for (i=1; i<=n; i++){
    for(j=1; j<=n; j++){
        k=j+1;
        while(k<=n){
            writer.print("-");
            buf = (i-1)*n + j;
            writer.print(buf +" -");
            buf = (i-1)*n + k;
            writer.print(buf+" ");
            writer.println("0");
            k++;
        }
    }

}
```

La première boucle itère sur l'indice de la ligne, les deux autres boucles permettent d'itérer sur les indices des deux cases à comparer. Chaque itération de la dernière boucle va donc créer une ligne du fichier au format DIMACS.
La partie concernant les colonnes est assez similaire au programme pour les lignes.

Pour les diagonales descendantes (le code pour les diagonales montantes est très similaire) :

```
// Part for the biggest diagonal
for (i=1; i<=n-1; i++){
    for(j=i+1; j<=n; j++){
        writer.print("-");
        buf = i + (i-1)*n;
        writer.print(buf +" -");
        buf = j + (j-1)*n;
        writer.println(buf+" 0");
    }
}
// Part for the other diagonals
for(i=2; i<=n-1; i++){
    for(j=i; j<=n-1; j++){
        for(k=j+1; k<=n; k++){
            writer.print("-");
            buf = i + (j-i)*(n+1);
            writer.print(buf +" -");
            buf = i + (k-i)*(n+1);
            writer.println(buf+" 0");
        }
    }
    for(j=i; j<=n-1; j++){
        for(k=j+1; k<=n; k++){
            writer.print("-");
            buf = i + (j-i)*(n+1) + (i-1)*(n-1);
            writer.print(buf +" -");
            buf = i + (k-i)*(n+1) + (i-1)*(n-1);
            writer.println(buf+" 0");
        }
    }
}
```

Le code pour les diagonales ressemble à celui des lignes ordinaires à l'exception des calculs des indices des cases et l'ordre de traitement des diagonales.

Le programme complet sera rendu joint avec le rapport au format numérique.

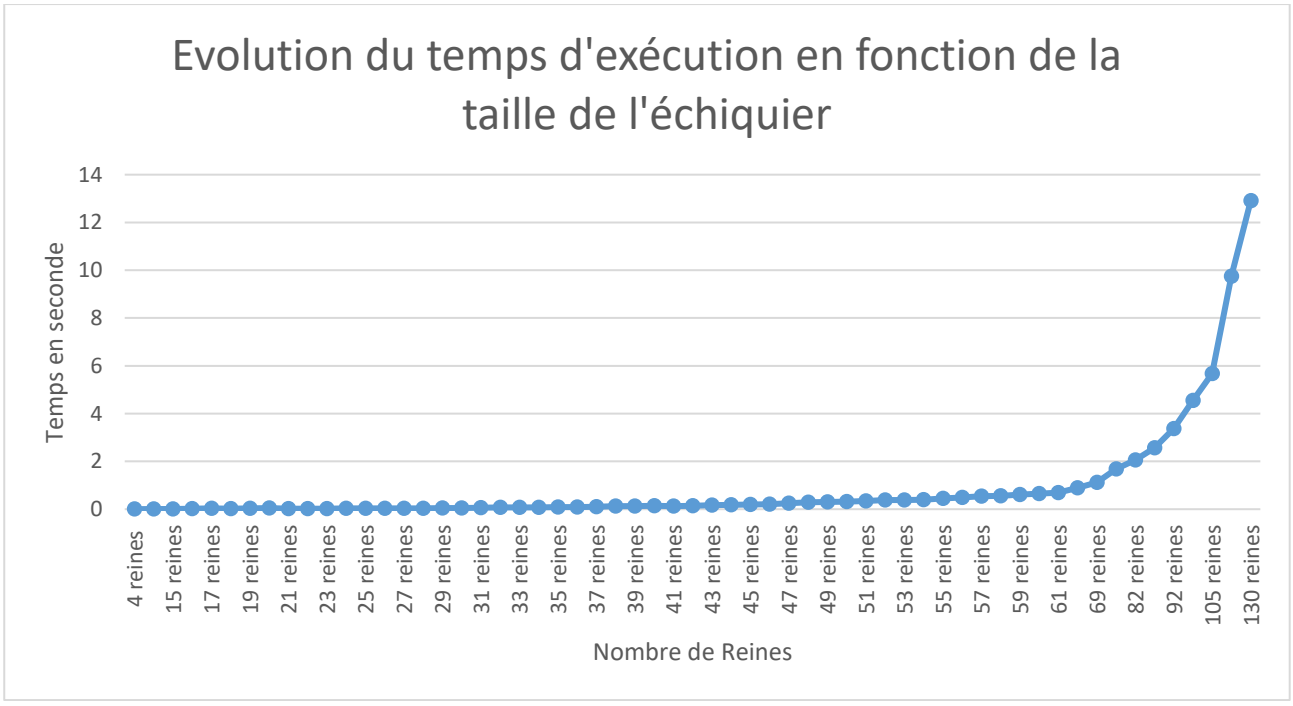
Expérimentation

Suite à cela, nous avons établi une plage de tailles de plateau, allant de 1*1 à 130*130, pour lesquelles nous allons déterminer la satisfaisabilité.

Ainsi, nous avons les résultats suivants :

Les valeurs pour lesquelles "n" n'est pas satisfaisable sont 2 et 3.

En fin d'expérimentation, nous avons augmenté la plage de valeurs pour la taille du damier afin d'obtenir des résultats plus significatifs. Le constat fait est que le temps d'exécution croît de manière exponentielle par rapport à la taille du damier.



Graphique 1: Evolution du temps d'exécution en fonction de la taille de l'échiquier

Conclusion

Cette expérimentation nous a permis de voir un exemple de procédure de décision. En effet, sans pouvoir déterminer exactement quelles étaient les solutions, nous avons pu déterminer les cas qui n'en possédaient aucune. Ainsi, nous avons pu répondre à la question de la satisfaisabilité du problème des n-reines, tout en nous familiarisant avec un solveur SAT et ses contraintes (format, limites, etc…). De plus, étant douze étudiants sur ce travail, cela nous a permis de faire une expérience de groupe supplémentaire.