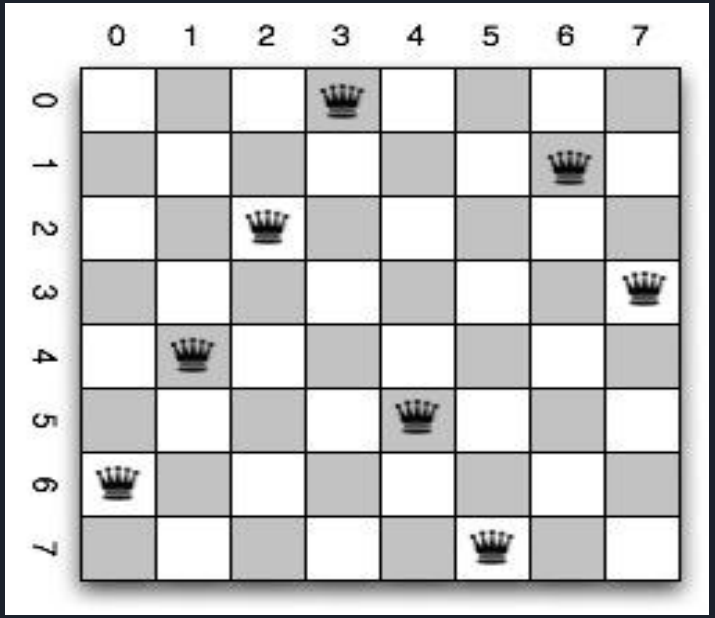
Projet:Modélisation et résolution

Matthew Coyle, Marwa Khadji, Amina El Haimour, Khaoula Abou-Bichara

Introduction Générale

Le problème de N-Reines



Modèle 1 : un modèle simple basé sur des variables ”domaine fini” entières (FD) sans contrainte globale .

- variables :

$X = \{ L_1, ..., L_n, C_1, ..., C_n \}$;

- domaines de définition :

$D(L_1) = ... = D(L_n) = D(C_1) = ... = D(C_n) = \{ 1..n \}$;

- contraintes :

Les reines doivent être sur des lignes différentes :

$\rightarrow Clig = L_i 6\neq L_j / i \in \{ 1..n \}, j \in \{ 1..n \}, i 6\neq j \}$

Les reines doivent être sur des colonnes différentes :

$\rightarrow Ccol = C_i 6\neq C_j / i \in \{ 1..n \}, j \in \{ 1..n \}, i 6\neq j \}$

Les reines doivent être sur des diagonales ascendantes différentes :

$\rightarrow Cdm = C_i + L_i 6\neq C_j + L_j / i \in \{ 1..n \}, j \in \{ 1..n \}, i 6\neq j \}$

Les reines doivent être sur des diagonales descendantes différentes :

$\rightarrow Cdd = C_i - L_i 6\neq C_j - L_j / i \in \{ 1..n \}, j \in \{ 1..n \}, i 6\neq j \}$

Modèle 1 : un modèle simple basé sur des variables ”domaine fini” entières (FD) sans contrainte globale .

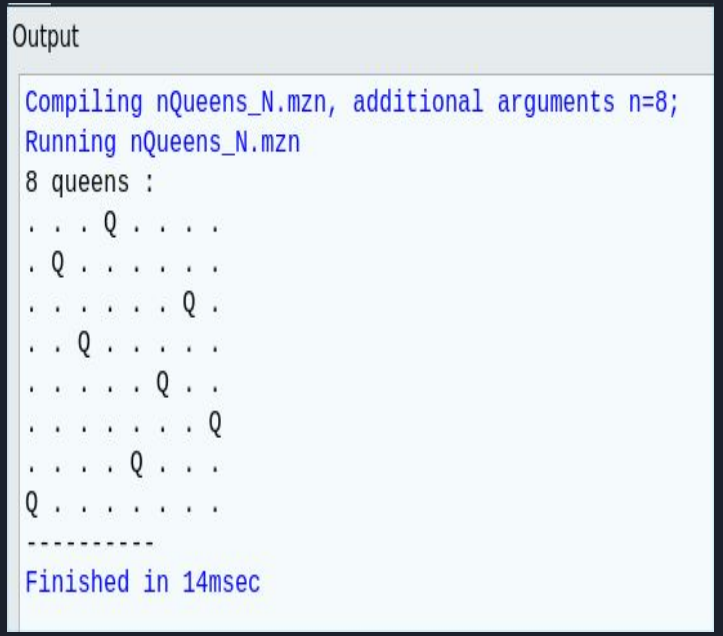
Implémentation :

- Par construction, aucune reine ne peut être sur la même ligne.
- (c[i] == c[j]).
- (abs(c[i] - c[j]) == abs(i - j)).

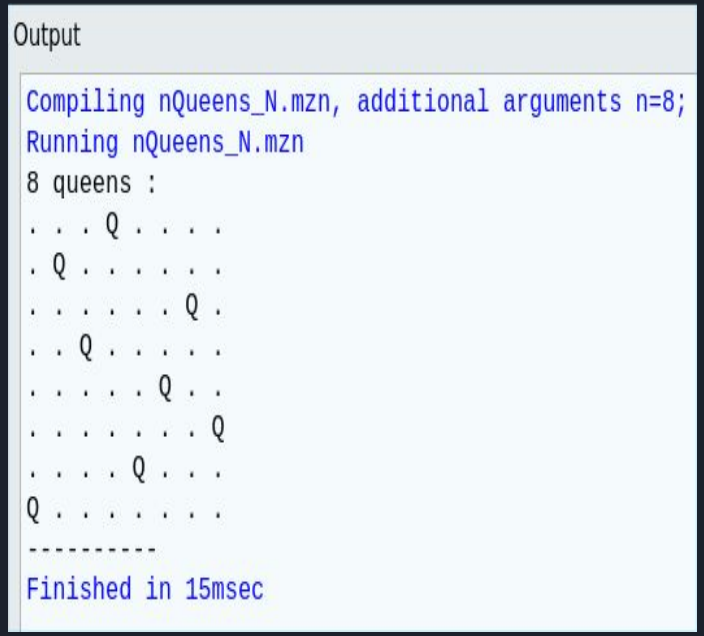
```
include "globals.mzn";
int:n;
% une seule reine par colonne
% nous pouvons concevoir cette contrainte dans la structure de données
% les reines sont des colonnes
array[1..n] of var 1..n: Q;
% pour chaque reine i, Q [i] nous donne la ligne
% pour les diagonales, on peut imaginer que les reines sont des points
% et l'échiquier est un système de coordonnées cartésien
% on veut éviter les pentes de + -1
constraint forall([
% la methode de la valeur abs
% abs(Q[j]-Q[i])!=abs(j-i)
% Notre proposition est plus optimale que la solution de la valeur absolue
 Q[i] != Q[j] /\ Q[i] + i != Q[j] + j /\ Q[i] - i != Q[j] - j
    |i,j in 1..n where i!=j
]);

solve satisfy;

output  [show(n)]++[" queens :\n"] ++
  [ if fix(Q[i]) = j then "Q " else ". " endif ++
    if j = n then "\n" else "" endif
  | i, j in 1..n
  ];
```

Résultat

Modèle 1:



Modèle Abs:



Modèle 2 : un modèle basé sur des variables ”domaine fini” entières (FD) avec contraintes globales et cassant des symétries

- La contrainte globale :
  allDifferent ($x_1, \ldots, x_n$)
- $x_i = l_i - i$
- $y_i = l_i + i1 \leq i \leq n$
- Contrainte pour supprimer la symétries verticale : Q[i1] <= n/2

Modèle 2 : un modèle basé sur des variables ”domaine fini” entières (FD) avec contraintes globales et cassant des symétries

```
include "globals.mzn";

int:n;
array[1..n] of var 1..n: Q;
% pour chaque reine i, Q [i] nous donne la ligne
% les lignes doivent être toutes différentes aussi c'est une contrainte globale

constraint alldifferent(Q);
constraint alldifferent(i in 1..n)(Q[i] + i);
constraint alldifferent(i in 1..n)(Q[i] - i);

% En cassant la symétrie
% cette contrainte supprime la symétrie verticale
constraint Q[1] <= n/2;

solve satisfy;
output  [show(n)]++[" queens :\n"] ++
  [ if fix(Q[i]) = j then "Q " else ". " endif ++
    if j = n then "\n" else "" endif
  | i, j in 1..n
  ];
```

Résultats

Output

```
Compiling M2_ACG_CS.mzn, additional arguments n=8;
Running M2_ACG_CS.mzn
8 queens :
Q . . . . . . .
. . . . . . Q .
. . . Q . . . .
. . . . . Q . .
. . . . . . . Q
. Q . . . . . .
. . . . Q . . .
. . Q . . . . .
----------
Finished in 25msec
```

Modèle 3 : un modèle basé sur des variables booléennes et contraintes booléennes

- variables :

$X = X_{ij}$, $i \in \{ 1..n \}$ etj $\in \{ 1..n \}$;

- domaines de définition :

$D (X_{ij}) = \{ 0, 1 \}$, $i \in \{ 1..n \}$, $j \in \{ 1..n \}$;

- contraintes :

Il y a une reine par ligne :

$\rightarrow Clig = \sum nj = 1 \; X_{ij} = 1/i \in 1..n$

Il y a une reine par colonne :

$\rightarrow Ccol = \sum i \; n = 1 \; X_{ij} = 1/j \in 1..n$

Les reines doivent être sur des diagonales ascendantes différentes :

$\rightarrow Cdm = i + j = k + l \Longrightarrow X_{ij} + X_{kl} \leq 1/i, j, k, l \in \{ 1..n \}$

Les reines doivent être sur des diagonales descendantes différentes :

$\rightarrow Cdd = i - j = k - l \Longrightarrow X_{ij} + X_{kl} \leq 1/i, j, k, l \in \{ 1..n \}$

Modèle 3 : un modèle basé sur des variables booléennes et contraintes booléennes

```
include "globals.mzn";
%-----------------------------------------------------------------------------%
% Parameters
int: n; % size
%-----------------------------------------------------------------------------%
% Variables
array[1..n,1..n] of var bool: board;
%-----------------------------------------------------------------------------%
%   x = bool2int(e) is equivalent to (e /\ x=1) \/ (not e /\ x=0).
predicate mul2dmor(array[int,int] of var bool: b, var int: m )=
  sum(i,j in 1..length(b))(bool2int(b[i,j]))==m;
% (b[i,j] /\ n=1) \/ (not b[i,j] /\ n=0))==m;

% Constraints
constraint forall(i, j in 1..n)(
  board[i,j] =  not (
    exists(j1 in 1..n where j1 != j)(board[i,j1])     %check row/col
  \/  exists(i1 in 1..n where i1 != i)(board[i1,j])   %check col/row
  \/  exists(k in 1..n-1)(                            %check diagonals
      board[i+k,j+k] \/ board[i-k,j+k]
      \/  board[i+k,j-k] \/ board[i-k,j-k]
    )
  )
);
constraint mul2dmor(board,n);                         %makes sure there are n queens

solve satisfy;
```

Resultat

```
Running M3_Boolean.mzn
board = array2d(1..8 ,1..8 ,[false, false, false, false, true, false, false, false, false, false, true, false, false, false, false, false, false, false, false, false, false, false,
false, true, false, false, false, true, false, false, false, false, false, false, false, false, false, false, true, false, true, false, false, false, false, false, false, false, false,
false, false, false, false, true, false, false, false, true, false, false, false, false, false, false]);

----------
Finished in 22msec
```

Modèle 4 : un modèle basé sur des variables booléennes et contraintes booléennes et cassant des symétries

- n(Q) est une variable globale
- Contrainte Q1 < n+1- Qn
- si Q1=n+1-Qn et Q2=n+1-Qn-1 alors Q3< n+1-Qn-2, etc. Sûr d'éliminer complètement cette symétrie, il faudrait ajouter jusqu'à n/2 contraintes à la formulation, chacune avec une condition de plus que la précédente.

Modèle 4 : un modèle basé sur des variables booléennes et contraintes booléennes et cassant des symétries

```
%-----------------------------------------------------------------------------%
% Predicates for symmetry breaking

predicate rot_sqr_sym(array[int,int] of var int: x) =
  let {
    int: n = card(index_set_1of2(x)),
        int: n2 = card(index_set_2of2(x)),
        int: l = n * n,
        array[1..l] of var int: y = [x[i,j] | i in index_set_1of2(x),
                                              j in index_set_2of2(x) ],
        array[1..4,1..l] of 1..l: p = array2d(1..4,1..l,
             [ if k == 1 then i*n + j - n else
               if k == 2 then (n - j)*n + i else
               if k == 3 then (n - i)*n + (n - j)+1 else
                              i*n + (n - j) - n + 1 endif endif endif
             | k in 1..4, i,j in 1..n ])
  } in assert(n == n2, "rot_sqr_sym: rotation symmetry applied to" ++
               " non square matrix",
    var_perm_sym(y,p));

predicate var_perm_sym(array[int] of var int: x, array[int,int] of int: p) =
   let { int: l = min(index_set_1of2(p)),
         int: u = max(index_set_1of2(p)),
         array[1..length(x)] of var int: y = [ x[i] | i in index_set(x)] } in
   forall (i in l..u, j in l..u where i != j) (
      var_perm_sym_pairwise(y, %% forces index 1..length(x)
                            [ p[i,k] | k in index_set_2of2(p)],
                            [ p[j,k] | k in index_set_2of2(p)]) );

predicate var_perm_sym_pairwise(array[int] of var int: x,
```

Modéle 3 avec MiniSat (format DIMACS)

Le programme génère les fichiers:

- clauses.txt: avec uniquement les clauses CNF-SAT DIMACS.
- code.cnf: avec la spécification CNF-SAT DIMACS complète.
- solution.txt : avec la relecture de la solution.

Modéle 3 avec MiniSat (format DIMACS)

```
p cnf 36 296
1 2 3 4 5 6  0
7 8 9 10 11 12  0
13 14 15 16 17 18  0
19 20 21 22 23 24  0
25 26 27 28 29 30  0
31 32 33 34 35 36  0
-1 -2 0
-1 -3 0
-1 -4 0
-1 -5 0
-1 -6 0
-2 -3 0
-2 -4 0
-2 -5 0
-2 -6 0
-3 -4 0
-3 -5 0
-3 -6 0
-4 -5 0
-4 -6 0
-5 -6 0
-7 -8 0
-7 -9 0
-7 -10 0
-7 -11 0
-7 -12 0
-8 -9 0
-8 -10 0
-8 -11 0
-8 -12 0
-9 -10 0
-9 -11 0
-9 -12 0
-10 -11 0
-10 -12 0
-11 -12 0
-13 -14 0
-13 -15 0
-13 -16 0
-13 -17 0
-13 -18 0
-14 -15 0
-14 -16 0
-14 -17 0
-14 -18 0
```

Résultat


etudiant@dCV5: ~/Desktop/Cours/Minisat/sat-nqueens
etudiant@dCV5:~/Desktop/Cours/Minisat/sat-nqueens$ make compile N_QUEENS=8
clang -o encoder encoder.c -DN=8
etudiant@dCV5:~/Desktop/Cours/Minisat/sat-nqueens$ time make run
./encoder
. . . . Q . . .
. . Q . . . . .
Q . . . .
. . . . . . Q .
. Q . . . . . .
. . . . Q
. . . . . Q . .
. . . Q . . . .
real 0m0,017s
user 0m0,012s
sys 0m0,006s
etudiant@dCV5:~/Desktop/Cours/Minisat/sat-nqueens$

Résultats et Discussion

	Modèle 1	Modèle 2	Modèle 3	Modèle 4
Solveur Gecode	n=8->13 ms n=128->2 m 24 s	n=8-> 15 ms n=128->48 ms	n=8-> 21 ms n=128-> +∞	n=8: 32ms n=128: +∞
Solveur Chuffed	n=8-> 128 ms n=128-> +∞	n=8 -> 158 ms n=128-> 1s 128 ms	n=8-> 206 ms n=128-> +∞	n=8: 192 ms n=128: +∞
Solveur SAT			n=8: 6 s n=128: 159 s	n=8: n=128:

Conclusion