

OLIVIER BAILLEUX

Université de Bourgogne - Master Informatique $2^{\text{ème}}$ Année BDIA



OLIVIER BAILLEUX PROJET PPC

1 Principe

L'objectif est de réaliser une pré-génération, ce développement s'est découpé en deux phases, un développement de découverte et mise en place des fonctionnalités, puis une refonte du code pour optimiser l'algorithme.

1.1 Phase 1 - Mise en place des fonctionnalités

Dans un premier temps, implémentation d'un tirage au hasard k fois en vérifiant simplement si la case n'était pas déjà utilisée. Suite à cela ajout de la vérification des diagonales.

Afin d'éviter que l'algorithme tourne indéfiniment ajout d'une limite du nombre de tentative avant que le générateur de s'arrête.

Répétition jusqu'à ce que k soit atteint :

Tirage au sort d'un X et d'un Y.

Vérification des diagonales.

Si pas de problème lors de la vérification des diagonales :

Enregistrement de X et Y dans init.

Remise a zero du nombre de tentatives.

Sinon:

Incrémentation du nombre de tentatives.

Si nombre de tentatives trop grand:

Arrêt du générateur.

1.2 Phase 2 - Optimisation

Dans un second temps, ajout des listes contenant les lignes et colonnes n'ayant pas encore de reines et ainsi éviter une grosse redondance dans le tirage de la reine à placer. Changement de la vérification des diagonales pour rendre le tout plus compacte et s'arrêtant plus rapidement en cas de conflit.

Ces modifications ont permis d'améliorer les performances sans toucher au nombre de tentatives avant l'arrêt du générateur.

Génération des listes des lignes et colonnes possibles dans pX et pY.

Répétition jusqu'à ce que k soit atteint :

Tirage au sort d'un X et d'un Y dans pX, pY.

Vérification des diagonales.

Si pas de problème lors de la vérification des diagonales :

Enregistrement de X et Y dans init.

Suppression de X et Y dans pX et pY.

Remise a zero du nombre de tentatives.

Sinon

Incrémentation du nombre de tentatives.

Si nombre de tentative trop grand :

Arrêt du générateur.

2 Structures de données

int[] init Structure fonctionnant comme IntVar[] vars de choco, stoquant les X et Y de la manière suivante : $init[x] \rightarrow y + 1$.

ArrayList<Integer> pX, pY Structure stockant les lignes et colonnes sans reines. Les X vont de 0 à n-1 et les Y vont de 1 à n pour correspondre aux structures init et vars.

int [] diag Structure stockant les 4 directions des diagonales, permettant de vérifier toutes les directions avec une simple boucle.

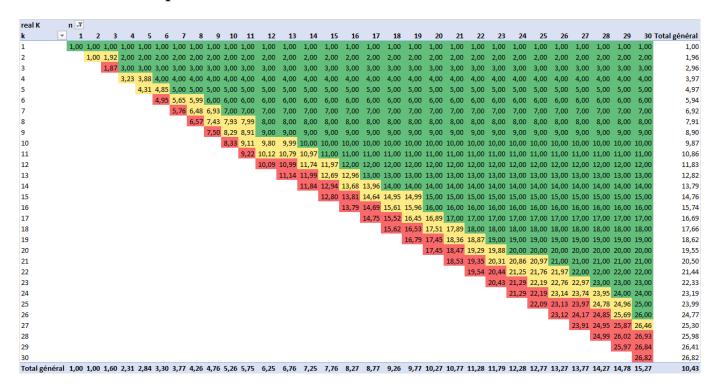
PROJET PPC OLIVIER BAILLEUX

3 Algorithmes

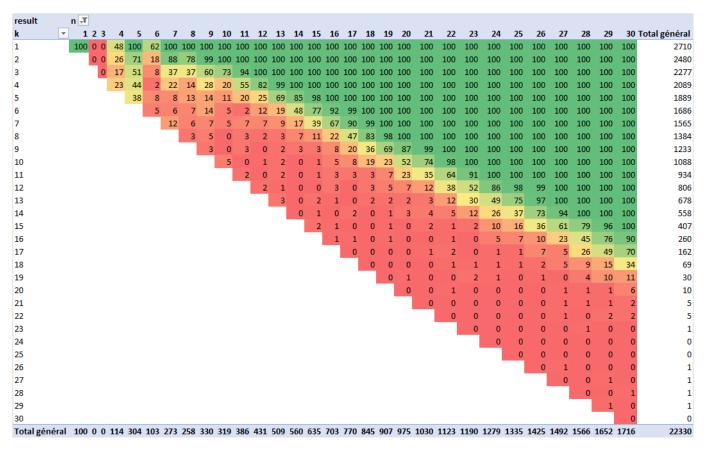
```
ALGORITHME generate
// Générateur maison qui place k reines sur les n
VARIABLES
    chk, isValid : BOOLEAN;
    iTry, x, y, dx, dy, cpt, nb : INTEGER;
    init : INTEGER[];
    pX, pY : ARRAYLIST<INTEGER>;
DEBUT
    pX <- gpX; //Liste des X (0..n-1)
    pY <- gpY; //Liste des X (1..n)
    init <- new INTEGER[n];</pre>
    iTry <- 0;
    nb <- 0;
    TANT QUE (nb < k) FAIRE
        x <- Tirage random de pX;
        y <- Tirage random de pY;
        chk <- VRAI;
        cpt <- 1;</pre>
        // Vérification des diagonales, temps qu'on a pas d'erreur
        TANT QUE (chk ET is Valid ET nb > 0) FAIRE
            chk <- FAUX;
            // On boucle sur les 4 directions
            POUR i ALLANT DE O A 4 FAIRE
                 dX <- diag[i][0] * cpt;</pre>
                 dY <- diag[i][1] * cpt;</pre>
                 // Vérification globale pour savoir si on est dans la grille
                 SI (x + dx \ge 0 ET x + dx < n ET y + dy > 0 ET y + dy < n + 1) ALORS
                     chk <- VRAI;
                     SI () ALORS
                         isValid <- FAUX;</pre>
                         STOP;
                     FIN SI
                 FIN SI
            FIN POUR
            ctp <- cpt + 1;
        FIN TANT QUE
        SI isValid ALORS
            init[x] <- y;
            // On retire le X/Y utilisé
            pX.REMOVE(x);
            pY.REMOVE(y);
        SINON
            SI (iTry > n * 1000) ALORS
                ARRET;
            SINON
                 iTry <- iTry + 1;
            FIN SI
        FIN SI
    FIN TANT QUE
FIN
```

OLIVIER BAILLEUX PROJET PPC

4 Résultats expérimentaux

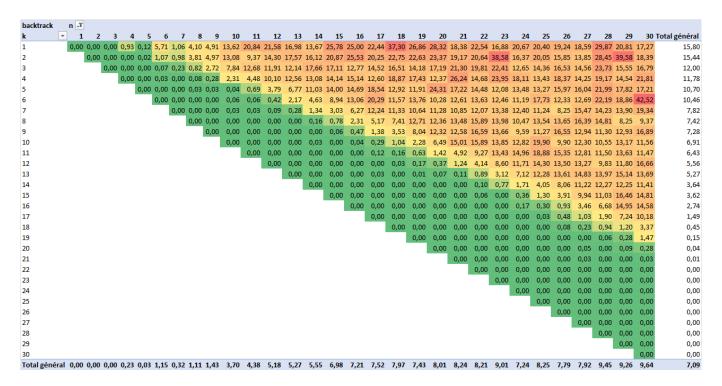


On peut voir que l'algorithme, pour N=30 arrive à placer toutes les reines jusqu'à K=N-4, entre K=N-4 et K=N-3 il manque 1 reine. Plus K tend vers N, plus le placement d'une reine au hasard est difficile. Plus N est grand, plus le problème apparait tôt.



Le solveur arrive, pour N=30, à trouver des solutions pour K < N/2 et en trouve un certain nombre pour 2N/3 < K < N/2. Plus N est grand, plus le solveur arrouve a trouver des solutions pour un K grand.

PROJET PPC OLIVIER BAILLEUX



Globalement, le nombre de backtrack est bien inférieur lorsque l'on place des points. Ce nombre trouve un minima vers K = N/2. Le solveur dit 0 lorsque celui-ci ne trouve pas de solution.

5 Sources

L'intégralité de mon code source et mes résultats all ant jusqu'à N=80 (fichier de sortie et images) sont disponibles sur mon Git Hub : https://git.io/vNtHi