Laboratoire 4
Algorithmes Génétiques

ELE440 - Algorithmes

Automne 2015

1 Objectifs

Ce laboratoire porte sur les algorithmes génétiques, un des principaux types de métaheuristiques utilisés pour résoudre des problèmes d'optimisation. Le but du laboratoire est :

1. de se familiariser avec les problèmes de recherche combinatoire ;
2. de se familiariser avec les problèmes NP-complets ;
3. de se familiariser avec le concept d'algorithme non-déterministe, comme les algorithmes génétiques ;
4. de concevoir certains des principaux blocs d'un algorithme génétique pour résoudre un problème de recherche combinatoire.

2 Implémentation

Écrire un programme qui permet de résoudre le problème des N reines à l'aide d'un algorithme génétique. Le problème consiste à déterminer une configuration de N reines sur un échiquier de $N \times N$ cases telle que les reines ne s'attaquent pas mutuellement. Le travail à faire est décrit dans les sous-sections suivantes.

Note importante : Des points seront alloués à la qualité de l'algorithme en termes de son agencement par rapport au problème particulier des N reines. Prenez le temps d'analyser le problème afin de bien diriger la conception de votre algorithme. Ne favorisez pas forcément la simplicité au détriment de la qualité.

2.1 Attaques possibles pour une reine

Les cases qu'une reine peut d'attaquer à partir d'une position donnée sont déterminées par les règles du jeu d'échecs. Une reine peut ainsi attaquer :

— Toute pièce située sur la même ligne qu'elle ;
— Toute pièce située sur la même colonne qu'elle ;
— Toute pièce située sur l'une de ses deux diagonales.

La figure 1 donne un exemple des cases qu'une reine peut attaquer partir de sa position sur un échiquier.

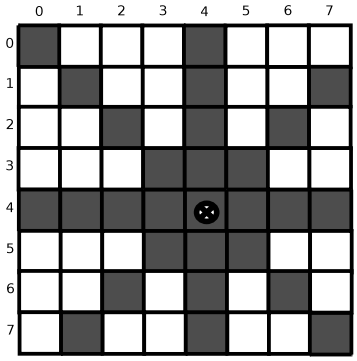


FIGURE 1 – Exemple des cases qu'une reine peut attaquer.

2.2 Encodage des solutions

L'encodage à suivre est donné ci-dessous. Le phénotype correspond à une configuration de N reines sur un échiquier de $N \times N$ cases.

L'encodage choisi pour le génotype est le suivant. Chaque chromosome est un tableau d'entiers de taille N **Chrom[0,...,N-1]**. Chaque position dans **Chrom** correspond à une colonne de l'échiquier et le nombre entier affecté à cette position indique la ligne de l'échiquier sur laquelle la reine située sur cette colonne est placée dans la configuration associée à la solution.
Cet encodage résout déjà une partie du problème puisqu'il exclut toute situation où deux reines occupent la même colonne.

Attention :

— Pour qu'une solution soit valide il faut que N reines soient présentes ;
— Deux reines ne peuvent pas occuper la même place sur l'échiquier.

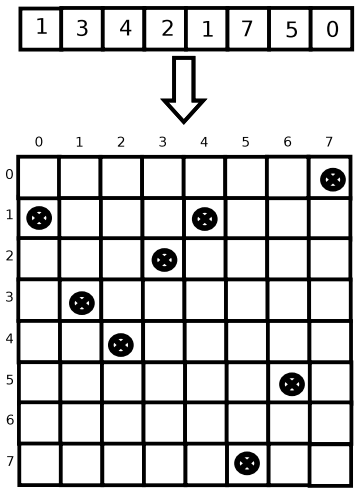


FIGURE 2 – Exemple illustrant l'encodage choisi.

2.3 Population initiale

La population initiale est générée aléatoirement ou lue dans un fichier. Dans le cas où elle est lue dans fichier, le fichier est organisé de la manière suivante :

— La première ligne indique la valeur de P, le nombre d'éléments (solutions) dans la population ;
— La seconde ligne indique la valeur de N, le nombre de reines (et la taille de l'échiquier) ;
— Les P lignes suivantes contiennent chacune N entiers indiquant la position de chaque reine sur l'échiquier.

2.4 Fonction de *fitness*

À partir d'une solution, on peut aisément calculer le nombre de conflits générés par une reine. On déduit ainsi facilement la pénalité associée à une solution en additionnant le nombre de conflits générés par toutes les reines. La fonction de *fitness* choisie pour notre problème est donc donnée par :

$$F(sol) = -\sum_{r=1}^{N} Conflit(r),$$

où $Conflit(r)$ indique le nombre de conflits générés par la reine r.

Le but du problème est de maximiser la fonction de *fitness* F jusqu'à la rencontre d'une ou plusieurs solutions pour lesquelles la fonction de *fitness* est égale à 0 (solution optimale puisqu'elle ne contient aucun conflit). Le pseudocode de la fonction calculant le nombre de conflits générés par une reine est donné en Annexe.

2.5 Opérateur de recombinaison

Vous êtes responsables de concevoir le ou les opérateurs de recombinaison qui seront utilisés par l'algorithme génétique. Vous pouvez utiliser les opérateurs présentés en cours (ou dans d'autres sources) ou en inventer de nouveaux.
À chaque opérateur de recombinaison *rec* est associée une probabilité α_{rec} de recombinaison pour deux individus de la population. Le pseudocode ci-dessous montre comment pourrait être utilisé le paramtre α_{rec} :

Pour chaque solution s1 dans la population
    Pour chaque solution s2 différente de s1 dans la population
        proba = rand(0,1) // ***Génère aléatoirement un nombre réel entre 0 et 1***
        Si proba $\leq \alpha_{rec}$
            Appliquer l'oprateur *rec* **à s1 et s2**

2.6 Opérateur de mutation

Vous êtes également responsables de concevoir un ou plusieurs opérateurs de mutation. Vous pouvez utiliser les opérateurs proposés en cours (ou dans d'autres sources) ou en inventer de nouveaux. Chaque opérateur de mutation est associé à un paramètre β_{mut} indiquant la probabilité de muter une solution avec l'opérateur de mutation *mut*.

Attention : Seuls les chromosomes enfants peuvent subir une mutation.
Le pseudocode ci-dessous montre comment pourrait être utilisé le paramètre β_{mut} :

Pour toute solution e dans l'ensemble des enfants
        proba = rand(0,1) // ***Génère aléatoirement un nombre réel entre 0 et 1***
        Si proba $\leq \beta_{mut}$
            Appliquer l'oprateur *mut* **à e**

2.7 Gestion de la population et processus de sélection

La population est de taille stable et égale P. Ainsi, vous devez générer P solutions afin de créer la population initiale et, à la fin de chaque itération, sélectionner P individus pour constituer la génération suivante.
La sélection des individus peut se faire selon le principe de la roulette biaisée ou par tournois. Vous pouvez inclure des règles de sélection supplémentaires, comme interdire les doublons, si vous le désirez. Dans tous les cas, il faut vous assurer de conserver la meilleure solution de la population.

2.8 Condition d'arrêt

L'arrêt de l'algorithme génétique est contrôlé par un paramètre G indiquant le nombre maximal de générations à étudier.

2.9 Paramétrage de l'algorithme par l'utilisateur

À chaque exécutation de l'algorithme génétique, l'utilisateur doit pouvoir ajuster les paramètres suivants de l'algorithme :

— Le nombre de générations ;
— La probabilité de sélection de chaque opérateur de recombinaison ;
— La probabilité de sélection de chaque opérateur de mutation.

Le mécanisme d'ajustement (entrée via une interface usager, via un fichier, etc.) est laissé à votre discrétion.

2.10 Renvoi des informations à l'utilisateur

À chaque exécution de l'algorithme génétique, vous devez exporter les informations suivantes à l'écran et dans un fichier :

— Le nombre de reines ;
— La taille de la population ;
— Le nombre de générations ;
— La probabilité de sélection de chaque opérateur de recombinaison ;
— La probabilité de sélection de chaque opérateur de mutation ;
— Le *fitness* de la meilleure solution obtenue ;
— Le temps de calcul de l'algorithme.

Si l'algorithme a trouvé des solutions optimales, votre programme doit les répertorier dans un second fichier en respectant le format suivant :

— La première ligne indique la valeur de P_{opt}, le nombre de solutions optimales trouvées ;
— La seconde ligne indique la valeur de N, le nombre de reines ;
— Les P_{opt} lignes suivantes contiennent chacune N entiers indiquant la position de chaque reine sur l'échiquier.

3 Livrables

Il n'y a pas de rapport écrit à remettre pour ce laboratoire, ce qui permettra une meilleure étude en vue de l'examen final. Le rapport prendra plutôt la forme d'une **présentation de 5 à 7 minutes** lors de la démonstration de votre programme au chargé de laboratoire. Cette présentation doit résumer le fonctionnement de votre algorithme génétique, décrire et justifier vos choix particuliers de conception sur les plans suivants (pas nécessairement dans cet ordre) :

— l'encodage des solutions (si des modifications ou des méthodes de réparation ont été introduites)
— le ou les opérateurs de recombinaison choisi(s)
— le ou les opérateurs de mutation choisi(s)
— l'approche de sélection choisie

Si vous proposez de nouvelles approches, **justifiez** vos choix en commentant sur les intuitions suivies. La présentation peut (ou non) utiliser un support visuel (ex : diapositives PowerPoint). Vous devrez également être en mesure de répondre á toute question que le chargé de laboratoire jugera pertinente á propos de votre algorithme.

De plus, vous devez remettre d'ici le **vendredi, 11 décembre à 23h59** votre **code source** (qui devra être adéquatement commenté pour permettre au chargé de laboratoire de comprendre vos choix de conception) et le tester sur une batterie de fichiers d'entrée qui seront fournis sur le site web du cours. Ces fichiers d'entrée correspondront à des problèmes et populations de différentes tailles. Vous devez remettre les résultats obtenus sur 5 essais différents avec votre algorithme sur ces cas de test sous la forme de **fichiers de sortie** correspondant à ces fichiers d'entrée avec votre code source. Il est conseillé d'écrire un programme ou un script pour exécuter automatiquement cette tâche d'expérimentation. Les programmes les plus performants dans ces tests et lors de la démonstration au chargé de laboratoire se mériteront plus de points.

La qualité de l'implémentation compte pour 80% de la note (cela inclut l'évaluation basée sur la performance de l'algorithme dans la batterie de tests à remettre le 12 décembre). La plus grosse partie de ces points seront alloués au fonctionnement de l'algorithme de base ; il est entendu que l'algorithme doit être en mesure de trouver des solutions optimales sur de petits problèmes (ex : $N = 8$), mais qu'il pourrait avoir plus de difficulté avec les problèmes de plus grande taille (ex : $N = 100$). Les programmes qui réussiront le mieux à résoudre les plus gros problèmes pourront se mériter davantage de points. Les 20% restant sont alloués à la présentation de l'algorithme et à la justification de vos choix de conception le soir de la démonstration.

3.1 Remarques finales

Ce laboratoire vous donne beaucoup de liberté sur la manière d'aborder le problème. L'énoncé vous indique les pistes fondamentales à suivre concernant les principaux blocs d'un algorithme génétique (encodage, opérateurs de recombinaison, opérateur de mutation, sélection) mais vous laisse beaucoup de latitude en termes de conception (la seule contrainte absolue concerne l'encodage qui vous force représenter les solutions sous forme de tableaux d'entiers de l'intervalle $[0, N-1]$). Les suggestions faites dans l'énoncé peuvent aisément être amliorées. Soyez créatifs, ce qui améliorera la qualité de votre algorithme et de vos notes.

Échéancier

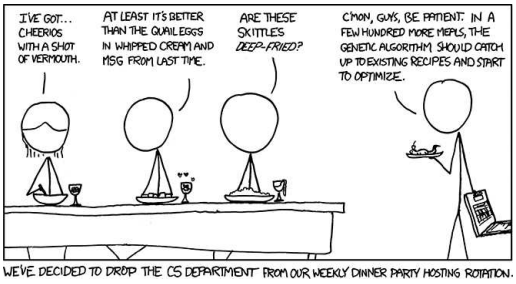
Semaine 1 :
- Analyse du problème
- Choix des opérateurs de recombinaison, mutation et sélection
- Lecture et génération des données
- Fonction de *fitness*

Semaine 2 :
- Implémentation des opérateurs de recombinaison, mutation et sélection
- Validation de l'algorithme génétique

Semaine 3 :
- **Démonstration du fonctionnement du programme au chargé de laboratoire**
- **Présentation de l'algorithme au chargé de laboratoire**
- Amélioration de l'approche

Vendredi, 11 décembre 2015 à 23h59 :
- Remise du code source et des résultats des tests.

ANNEXES



Source : http ://www.xkcd.com

Squelette de l'algorithme génétique à implémenter [Holand76] :

algoGen(P,G,α, β)

Pop=$\varnothing$
Pop = getPopulation (P) // Génération aléatoire ou importation à partir d'un fichier
Pour g allant de 1 G
listeParent = Pop ;
listeEnfant = Recombinaison(listeParent,α) ;
listeEnfantMut = Mutation (listeEnfant, β)
Pop = Selection(listeParent,listeEnfant,listeEnfantMut,P)

Algorithme calculant le nombre de conflits générés par une reine :

conflitReine (Chrom[0,...,N-1], indiceReine, positionReine)

nbConflits = 0

k = 1
Pour j = indiceReine+1 à N
Si (Chrom[j] == positionReine+k) //Conflit sur la diagonale bas-droite
nbConflits = nbConflits+1
Si (Chrom[j] == positionReine-k) //Conflit sur la diagonale haut-droite
nbConflits = nbConflits+1
Si (Chrom[j] == positionReine) //Conflit sur la ligne à droite
nbConflits = nbConflits+1
k = k+1
Retourner nbConflits