Algorithme Génétique et Gestion de Projet

Projet AGGP - Dossier de conception

Anthony Tschirhard, Marie Paturel, Marion Poirel, Balthazar Rouberol

INSA Lyon - 4BIM

30 mars 2011

L'objectif de ce document est de prévoir la conception du projet en explicitant l'organisation du code ainsi que le planning des phases de codage, de tests et d'exploitation du code. Ainsi, sa lecture devrait permettre de connaître précisément la division du projet, à la fois au niveau humain (organisation au sein de l'équipe) et temporel.

1 Présentation générale du projet

1.1 Algorithme génétique

Le but de ce projet est de faire évoluer une population de réseaux grâce à un algorithme génétique pré-existant. Nous devons définir une *fitness* qui permet de sélectionner les individus sur leurs capacités à s'approcher d'un réseau biologique.

1.2 Livrables et délais

Nous devons produire un réseau respectant les propriétés d'un réseau biologique, et pouvoir le présenter pour le mercredi 13 avril, c'est-à-dire 5 semaines après le début du projet.

1.3 L'équipe

- Chef de Projet : Balthazar Rouberol
- Responsable Qualité : Marion Poirel
- Maître d'œuvre : Anthony Tschirhard
- Responsable Documentation : Marie Paturel
- Groupe d'étude informatique : Balthazar Rouberol, Anthony Tschirhard, Marion Poirel, Marie Paturel

2 Propriétés des réseaux biologiques

2.1 Rappels

Rappelons ici les propriétés générales d'un réseau biologique à valider par les réseaux à générer :
- la répartition de ses degrés devra suivre une loi de puissance $P(k) \sim \alpha k^{-\gamma}$, traduisant la faible présence de nœuds ultraconnectés : les hubs
- validation de la propriété de « petit monde »
- formation de cliques

2.2 Tests et paramètres

Paramètres de la loi de puissance
La bibliographie suggère de fixer γ entre 2 et 3 [1] ou de le fixer à 2.1 [2]. Comme visible sur la figure 1, la différence de distribution est relativement faible entre les deux valeurs extrêmes. Nous décidons donc de fixer γ à 2.1.

1. Network biology : Understanding the cell's functional organization - A. Barbabási & Z. Oltvai - Nature reviews - Genetics - Feb 2004
2. Exploring complex networks - S. Strogatz - Nature Vol 410 - March 2001

Propriété de petit monde
Cette propriété traduit le fait que tous les nœuds d'un graphe sont reliés. La valeur de la longueur moyenne entre les nœuds dans un réseau métabolique est comprise entre 3 et 4[3]. Nous décidons donc de conserver cette condition et de l'appliquer aux réseaux générés.

Formation de cliques
Un graphe compte un certain nombre de cliques de différentes tailles. Une clique est un ensemble de nœuds entièrement interconnectés. Le coefficient de *clustering* est un bon indicateur de la présence de cliques et se calcule pour chaque nœud de la manière suivante :

$$C_i = \frac{2Ei}{k_i k_{i-1}}$$

avec
- E_i : nombre de liens entre voisins ;
- k_i : nombre de voisins de i.

Nous allons utiliser le coefficient de *clustering* global, c'est-à-dire la moyenne des coefficients de chaque nœud. Ainsi, nous aurons une valeur comprise entre 0 et 1 que nous pourrons utiliser dans notre fonction de *fitness*.

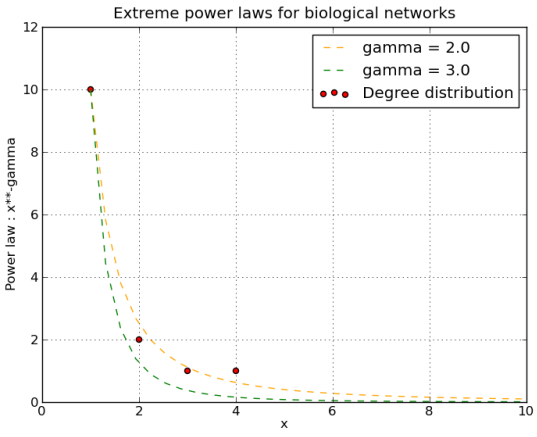

FIGURE 1 – Distribution puissances limites

3 Implémentation

3.1 Classes et objets

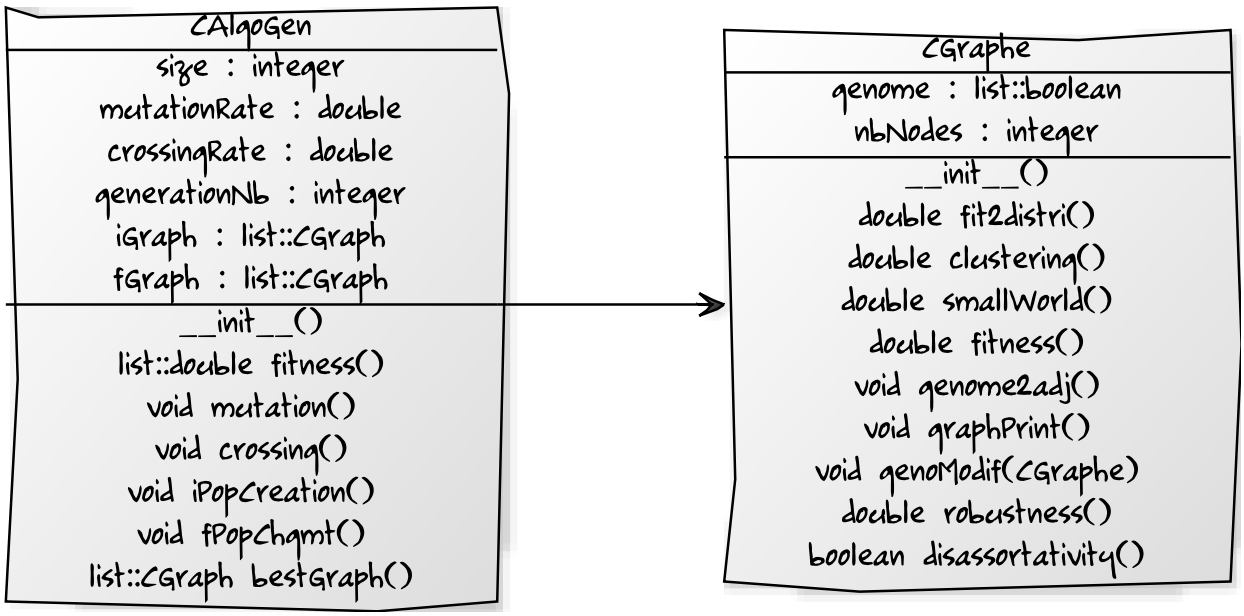

FIGURE 2 – Diagramme de classe

Comme le montre la figure 2, notre programme se compose de 2 classes principales : la classe **CGraph** et la classe **CAlgoGen**.

3. Network biology : Understanding the cell's functional organization - A. Barbabási & Z. Oltvai - Nature reviews - Genetics - Feb 2004

3.2 Codage d'un réseau

3.2.1 Matrice d'adjacence

Afin de coder nos réseaux, nous allons nous servir d'une matrice d'adjacence : une matrice de dimension $n * n$ dont l'élément non-diagonal a_{ij} est le nombre d'arêtes liant le sommet i au sommet j. L'élément diagonal a_{ii} est le nombre de boucles au sommet i.

Notre graphe est simplifié :
- graphe non orienté, *ie.* notre matrice est diagonale ($\forall i, a_{ij} = a_{ji}$),
- absence de boucle aux sommets, *ie.* la diagonale est nulle ($\forall i, a_{ii} = 0$)

3.2.2 Remplissage de la matrice

La matrice peut être générée, par `NetworkX` de deux manières : soit de manière aléatoire, soit de manière "biaisée", en influençant par exemple la répartition des degrés dans le graphe de départ. En effet, on veut converger vers nos trois critères de *fitness* mais ceux-ci ne sont pas tous aussi simples à atteindre. Il est par exemple assez simple de converger vers la petit monde alors que converger vers la répartition des degrés voulue est plus long.

Nous avons donc décidé de partir d'un réseau dont la répartition des degrés de départ est biaisée et suit une loi de puissance – celle voulue.

Nous nous servirons donc de la concaténation des lignes de la partie triangulaire supérieure de la matrice d'adjacence pour modéliser le "génome" d'un réseau.

3.2.3 Fonctions

fit2distri() Cette fonction permet de voir si notre répartition de degré correspond bien à la répartition théorique voulue (en utilisant la somme des carrés des écarts à la distribution attendue).

clustering() Cette fonction récupère simplement le coefficient de *clustering* renvoyé par `NetworkX`

smallWorld() Cette fonction récupère la distance moyenne entre deux nœuds renvoyée par `NetworkX`

fitness() Cette fonction calcule la *fitness* de notre réseau (cf. 3.4). La *fitness* renvoyée est comprise entre 0 et 1.

genome2adj() Cette fonction convertit notre génome – suite de 0 et de 1 – en matrice d'adjacence interprétable par la suite par `mat2graph()`.

mat2graph() Cette fonction fait la liaison entre notre matrice d'adjacence et `NetworkX`.

graphPrint() Cette fonction imprime notre graphe (avec légende etc.)

genoModif(CGraph) Lorsque qu'il est nécessaire que le graphe soit modifié, cette fonction permet le changement.

robustness() Cette fonction détermine la robustesse du graphe (cf. 3.3)

disassortativity() Cette fonction la propriété de non-assortativité des hubs (cf. 3.3)

3.3 Algorithme génétique

Nous utiliserons ici le module `AG.py` codée en scéances d'Optimisation.

À chaque génération, on calcule la *fitness* de chaque individu, qui est définie par une combinaison linéaire de coefficients témoignant de leur adéquation aux propriétés d'un réseau biologique (cf section 3.4). Les individus sont classés en fonction de leur *fitness*. On conserve les 10% meilleurs individus pour la nouvelle génération. Ceux-ci subissent des mutations, des croisements, étapes fondamentales de l'évolution.

Une fois la génération finale obtenue, on teste la robustesse du meilleur réseau obtenu (conservation des propriétés malgré la suppression au hasard d'un nœud) ainsi que leur concordance avec la propriété de non-assortativité [4] (les hubs ne sont pas reliés entre eux).

On lancera cette simulation plusieurs fois afin de mesurer le temps moyen de convergence. Les paramètres finaux des réseaux type « biologique »sont conservés afin de permettre un étude comparative. D'un simulation à une autre, nous feront varier la pondération des paramètres de la loi de *fitness* et le degré de la loi de puissance, ce qui nous permettra d'étudier leur impact respectif.

3.4 Fonction de *fitness*

La textitfitness d'un individu est un nombre réel compris entre 0 et 1, 1 correspondant à une textitfitness maximale. Celle-ci est calculée comme une somme linéaire pondérée des résultats renvoyés par les fonctions `clustering()`, `fit2distri()`, et `smallWorld()`, testant l'adéquation des du réseau aux paramètres principaux d'un réseau biologique : formation de cliques, distribution des degrés selon une loi de puissance et propriété de petit monde.

Nous pondérons cette somme pour favoriser un critère par rapport aux autres. En effet, la coexistence de propriété de petit monde et de formation de cliques est fondamentalement paradoxale. Sans ce favoritisme, l'émergence de réseaux biologique est fortement freinée, voir inhibée.

3.5 Évolution des réseaux

3.5.1 Mutation

On va faire évoluer nos réseaux en mutant certains individus pris au hasard dans la population. Ces mutations se caractériseront par des changements 0/1 au sein de la matrice d'adjacence. Il conviendra de déterminer avec logique la fréquence de ces mutations.

3.5.2 *Crossing-over*

Parmi les individus sélectionnés à chaque génération – ceux ayant la meilleur *fitness* – on va effectuer des crossing-over de manière aléatoire. Encore un fois, il faudra déterminer intelligemment leur taux et fréquence.

4. Structure of biological, Presentation by Atanas Kamburov, 08.05.2007

4 Planning

4.1 Répartition du travail

Fonction	Attribué à	Date de rendu
Adaptation de `AlgoGen`	Marie	03/04
`fit2distri()`	Balthazar	31/03
`clustering()`	Marie	01/04
`smallWorld()`	Marion	01/04
`fitness()`	Anthony	03/04
`genome2adj()`	Balthazar	31/03
`graphPrint()`	Balthazar	01/04
`genoModif()`	Anthony	03/04
`robustness()`	Marion	05/04
`dissasortativity()`	Marion	05/04

Chacun est responsable de tester les fonctions qui lui sont attribuées avant la date de rendu. Ces tests se feront sur des graphes créés avec `NetworX` (manuellement et de façon automatique : scale-free, aléatoire, etc.).

4.2 Exploitation

Nous commençons par tester notre code sur une population de 50 individus, chacun composé de 15 nœuds. Ceci nous permet d'approximer le temps de convergence et ainsi de mieux maîtriser les délais d'exploitation.

Nous pouvons alors trouver un compromis entre taille de la population et nombre de nœuds dans un réseau. A priori, une population de 50 individus ayant chacun 100 nœuds nous semble un bon début. Ces paramètres seront à adapter en fonction du temps de calcul.

Après avoir lancé une simulation, plusieurs seront lancées en parallèle, chacune ayant des paramètres différents : taille de la population, taille d'un réseau, pondération de la fonction de *fitness*, paramètre de la loi de puissance. Les résultats obtenus seront exploités en vue de déterminer leur influence respective sur la formation d'un réseau biologique.

4.3 Risques

Nous utilisons Python qui est un langage de haut niveau : le temps de calcul des fonctions proposées est a priori optimisé mais l'encapsulation ne permet pas sa maîtrise. De plus, nous utilisons pour la première fois le module `NetworkX`. Bien que nous ne le maîtrisions pas parfaitement, une partie de notre code repose entièrement dessus. Nous ne connaissons donc pas les performances calcul de ce module.

Il est donc à craindre une mauvaise appréhension du temps de calcul liée à la taille de la population et des réseaux ainsi qu'à l'encapsulation de nombreuses fonction `Python`.