Institut National des Siences Appliquées,
Bioinformatique et modélisation

Algorithmie génétique et Gestion de projet

Dossier de conception

Raphaëlle SAUZEDE, Cécile PIZOT, Samuel PICHOT
Adrien WAGNER et Florian THÖNI

Lyon, le 24 mars 2010

Table des matières

1 Présentation générale du projet

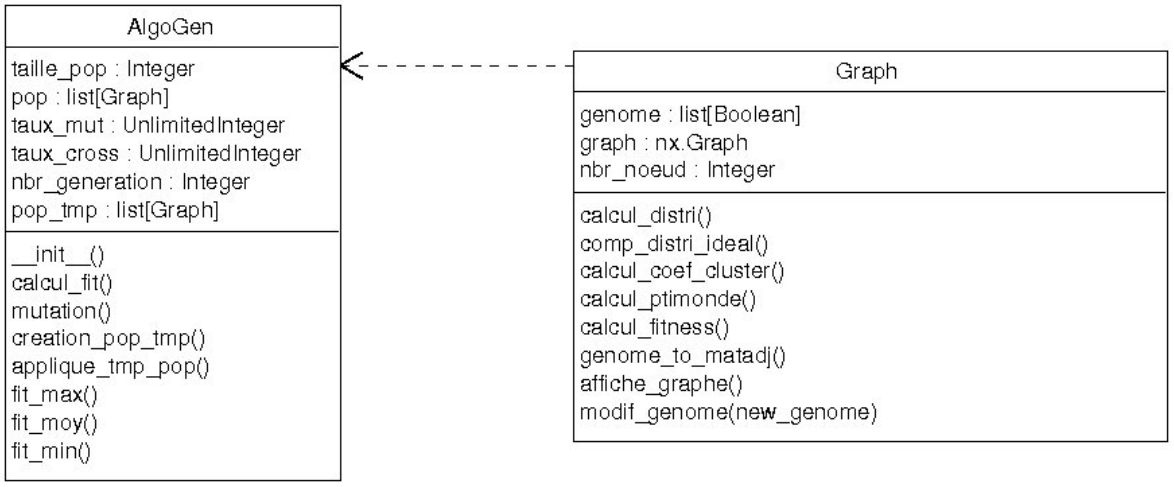
La réalisation d'un réseau biologique est un problème multi-contraintes que nous allons résoudre en utilisant un algorithme génétique. Ce type de réseaux respecte trois règles principales, à savoir, la propriété de petit monde, la formation de cliques et la répartition des degrés en loi de puissance, que nous devrons prendre en compte tout au long de notre projet.

La conception de notre programme repose sur trois étapes clés. Tout d'abord, il va nous falloir implémenter les trois propriétés essentielles des réseaux biologiques. Ensuite, nous allons utiliser ces différentes règles pour calculer une fitness pour chaque individu de la population. Enfin, nous allons répéter n fois les calculs de l'algorithme génétique afin d'obtenir une population se rapprochant de l'optimum c'est-à-dire du réseau biologiquement acceptable.

Le problème de création d'un réseau artificiel biologiquement pertinent sera résolu au travers d'une application python. Nous l'utiliserons nous-même pour créer un réseau de 500 noeuds maximum (pour des raisons de temps de calcul) que nous présenterons lors de la soutenance du projet qui aura lieu fin Avril.

2 Architecture du programme

2.1 Diagramme de classes



2.2 Classe Graphe

2.2.1 Attributs

La classe *Graphe* correspond à des individus, caractérisés par leur génome et contient trois attributs :

- *nbr_noeud* correspond au nombre de noeuds dans le graphe.
- *genome* est une liste contenant des 0 et des 1. Il correspond à un individu de la population. Il est construit à partir de la concaténation des lignes de la partie triangulaire supérieure de la matrice d'adjacence. Sa taille sera donc $\frac{NbrNoeud*(NbrNoeud-1)}{2}$.
- *graph* correspond à l'objet networkX.graph créé à partir du génome (matrice d'adjacence) à chaque modification de celui-ci.

2.2.2 Fonctions

La classe *Graphe* contient 5 fonctions liées aux caractéristiques de graphes et au calcul de la fitness ainsi que trois fonctions annexes.

Les fonctions centrales correspondant à l'implémentation des trois régles et au calcul de la fitness :

- *calcul_distri()* permet de calculer la loi de distribution du graphe. Elle renvoie une liste de taille nbr_noeuds avec pour chaque indice correspondant au degré, sa fréquence associée. Par exemple à l'indice i nous aurons la proportion de noeuds de degré i.
- *comp_distri_ideal()* compare la loi effective du graphe à la loi théorique qui est une loi de puissance. Elle renvoie une valeur réelle comprise entre 0 et 1.
- *calcul_coef_cluster()* permet de calculer le coefficient de clustering global du graphe et renvoie une valeur réelle comprise entre 0 et 1.
- *calcul_ptimonde()* permet de calculer le nombre de degré de séparation moyen entre les nœuds pris deux à deux. Elle renvoie une valeur comprise entre 0 et n.
- *calcul_fitness* permet de calculer la fitness de chaque individu en s'appuyant sur les trois règles. Nous proposons de rendre cette fitness bornée entre 0 et 1.

Les fonctions permettant la création d'un objet graphe sont les suivantes :

- *genome_to_matadj()* permet de passer du génome sous forme de vecteur au génome sous forme de matrice d'adjacence.
- *affiche_graphe()* permet la représentation graphique du graphe.
- *modif_genome(new_genome)* permet de remplacer le génome actuel par le nouveau génome rentré en paramètre.

2.3 Classe AlgoGen

2.3.1 Attributs

La classe *AlgoGen* contient six attributs :
- *taille_pop* est un entier correspondant à la taille de la population.
- *pop* est une liste d'instances de *Graph*, individus formant la population.
- *taux_mut* et *taux_cross* qui sont des réels représentant respectivement les taux de mutation et de crossing-over. Ils vont nous permettre de faire évoluer la population.
- *nbr_generation* est un entier définissant le nombre de fois que l'algorithme va être appliqué.
- *pop_tmp* est une liste d'individus permettant la création de la nouvelle population sans écraser l'ancienne le temps d'y appliquer les traitements nécessaires.

2.3.2 Fonctions

La classe *AlgoGen* contient huit fonctions :
- *__init__()* pour la génération d'une population initiale créée aléatoirement.
- *calcul_fit()* qui fait appel à la fonction *calcul_fitness()* de chaque individu pour calculer leur fitness respective.
- *mutation()* et *crossing_over()* permettent d'appliquer les traitements évolutifs à la pop_tmp.
- *creation_pop_tmp()* permet de créer une population temporaire en sélectionnant des individus parmi la population courante en les tirant aléatoirement sur une roulette biaisée.
- *applique_tmp_pop()* permet de remplacer la population actuelle par la population temporaire.
- *fit_moy* et *fit_min* et *fit_max* permettent de calculer respectivement la fitness moyenne, la fitness minimum et la fitness maximum de la population courante.

3 Implémentation des trois propriétés

La réalisation des trois fonctions nécessite de connaître les paramètres réels des réseaux biologiques.

3.1 Répartition des degrés : réseau scale-free

Un réseau scale-free possède une répartition des degrés en loi de puissance ($P(k) \sim \alpha k^{-g}$) c'est-à-dire qu'il possède des noeuds très connectés appelés hubs et une majorité de noeuds qui sont faiblement connectés. D'après les articles, le g doit-être compris entre 2 et 3, de manière plus précise entre 2.1 et 2.4, pour les réseaux biologiques. Nous choisissons arbitrairement que notre réseau idéal aura un g de 2.2. Nous allons nous appuyer sur ces deux conditions pour implémenter la fonction.

Nous allons parcourir l'ensemble des noeuds du graphe et calculer la fréquence pour chaque degré. Ensuite, nous comparons cette distribution à une loi de puissance en calculant la distance au carré entre la valeur réelle et la valeur théorique.

3.2 Formation de cliques : coefficient de clustering

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 ainsi pour chaque noeud :

$$C_i = \frac{2 \cdot E \cdot i}{k_i \cdot (k_i - 1)}$$

Avec E_i le nombre de liens entre voisins et k_i le nombre de voisins de i.

Nous utiliserons le coefficient de clustering global qui correspond à la moyenne des coefficients locaux, c'est à dire les C_i. La valeur renvoyée est comprise entre 0 et 1 (1 correspond à une abondance de cliques).

3.3 Propriété de petit monde

La propriété de petit monde nous indique que deux noeuds pris au hasard dans le réseau peuvent être reliés par un chemin comprenant seulement quelques arêtes. Nous n'avons pas de valeur de la distance moyenne entre deux nœuds pour les réseaux biologiques. Nous allons donc calculer cette distance sur quelques réseaux biologiques réels pour avoir un ordre de grandeur de la valeur recherchée.

Nous utiliserons une fonction du package NetworkX qui nous fournira directement la distance moyenne entre deux noeuds que nous comparerons par la suite à la valeur trouvée dans les réseaux biologiques.

4 Implémentation d'une fonction de Fitness

Nous souhaitons avoir une fitness majorée comprise entre 0 et 1. Nous utiliserons donc pour la calculer la moyenne de 3 valeurs issues plus ou moins directement des trois propriétés.

La valeur renvoyée par la fonction *comp_distri_ideale()* est comprise entre 0 et 1, il en va de même pour le résultat du coefficient de clustering.

En revanche la fonction qui calcule la distance moyenne entre deux noeuds renvoie une valeur entre 0 et nbr_noeuds. Nous allons la transformer pour obtenir un résultat entre 0 et 1, 1 représentant le cas idéal :

$$Indicateur = 1 - \frac{|D_{reel} - D_{observe}|}{D_{reel}}$$

Avec D_{reel} la distance moyenne dans les réseaux biologiques et $D_{observe}$ la distance moyenne observée dans notre réseau.
Ensuite la moyenne de ces trois valeurs est réalisée pour obtenir la fitness.

5 Répartition des tâches au sein du groupe

- *Finalisation et optimisation de l'algorithme génétique* 7 avril Florian
- *Fonction génome to Matrice d'adjacence* 7 avril Samuel
- *Fonction créant un réseau network X à partir de la matrice d'adjacence* 7 avril Cécile
- *Implémentation de la règle 1* 7 avril Adrien
- *Implémentation de la règle 2* 7 avril Raphaëlle, Cécile
- *Implémentation de la règle 3* 7 avril Raphaëlle, Adrien
- *Implémentation de la fonction de calcul de la fitness* 7 avril Raphaëlle, Samuel
- *Calculs et sélection du graphe* 14 Avril Par binômes
- *Soutenance* Fin avril