Rapport De Projet De Programmation

Pierre IMBERT Maxime KERMARREC

6 avril 2019

I) Compilation et éxécution

Pour la compilation faire make

Pour l'exécution faire ./yersin

II) Implémentations et extensions

II.1) Les implémentations faites pour ce projet.

1. Clique
2. Anneau
3. Erdos-Renyi
4. Chargement depuis un fichier texte format stanford : gestion des sommets manquants et allocation d'un tableau de taille n et non nmax.
5. Ecriture dans un fichier texte format stanford
6. La Prairie aléatoire (graphe de disques aléatoire)
7. Small World aléatoire (anneaux de Watts–Strogatz)
8. Réseau social aléatoire (modèle de Barabási–Albert)

II.2) Quelques extensions personnelles :

1. Mise en place d'une politique vaccinale
2. La mort. En plus des trois états S, I et R nous en avons rajouté un quatrième qui est la mort (M).
3. Pour le modèle de prairie aléatoire, il est possible de modifier le rayon (diminution ou augmentation) après la mort d'un individu ce qui entraîne une mise à jour des connexions.

III) Caractéristiques liées au programme

Lors de nos différents essais nous n'avons pu constater aucun : warnings à la compilation, crash à cause de nos options ou données, de fichier non fermé ou encore de fuite mémoire. De plus afin de vérifier si nous avions des fuites mémoires, nous avons effectué la commande suivante :

valgrind –tool=memcheck –leak-check=full –leak-resolution=high –show-reachable=yes –track-origins=yes ./yersin

Et nous n'avons constaté aucune fuite.

IV) Transitions G et D

Pour les transitions G et D, deux possibilités nous sont présentées l'une est temporelle l'autre probabiliste. Or nous avons implémenté les deux. En effet, épidémiologiquement parlant il est plus cohérent que plus les jours passent plus l'on a de chance de guérir d'une maladie. Or il existe une variabilité intrinsèque à chaque individu, selon ses propres défenses immunitaires. Ainsi dans notre modèle il existe une probabilité basique de passer d'un état à un autre, donnée en ligne de commande, qui augmentera à chaque tours selon le nombre de tours passé dans un état pour chaque individu.

V) Travail fait

Tous les paramètres dont nous avons besoin afin de lancer le programme sont donnés dans un fichier que nous ouvrons, lisons et fermons avant le lancement de la simulation.

a) État initial

Tous les individus de départ sont sains, sauf un nombre donné dans le fichier, qui sont tirés au sort au début du programme et seront infectés. Il peut aussi y avoir un nombre d'individu initialement résistant séléctionné également aléatoirement.

b) Topologie

Toutes les topologies du projet2.pdf ont été implémentées.

c) Condition d'arrêt

Le programme tourne au plus un certain nombre de tours et s'arrête avant si stabilisation, c'est-à-dire qu'il n'y a plus d'infecté et les morts ne peuvent plus transmettre la maladie (les morts restent infectueux pendant une période de 10 jours).

d) Résultat

Lorsqu'on réalise un malloc, calloc ou réalloc en cas d'erreur on met fin au programme. Cela est également valable pour l'ouverture des fichiers et les scanf lorsque nous demandons un integer (arrêt si un char est renseigné). Lorsqu'un fichier de configuration est chargé nous affichons les différentes données de la simulation avant son lancement. Pendant la simulation nous affichons le tour actuel ainsi que le nombre de S, I, R et M à chaque tour. A la fin de la simulation un rapide résumé affiche la vitesse de propagation moyenne de l'épidémie ainsi que le degré moyen. L'utilisateur peut également choisir d'enregistrer les résultats dans un fichier texte dont-il renseigne le nom dans le programme (scanf). Ce fichier indique pour chaque utilisateur son nombre de voisin ainsi que ces voisins au début de la simulation. Pendant la simulaion, à chaque tour est également renseigné l'état dans lequel se trouve chacun des individus. La liste des infectés, des résistants et des morts s'affiche également. Renseigné /dev/null empêche l'écriture.

e) Détail des implémentations

De plus, pour tous nos modèles nous générons un fichier de sortie au format de Stanford.

Modèle d'Erdős–Rényi :

Après avoir généré le graphe, nous le stockons dans un fichier. Ce fichier a un nom paramétrable et est émis dans le répertoire courant.

Chargement depuis un fichier texte:

– On utilise un tableau pour les individus et des listes chaînées pour les voisin, il faut une première passe pour déterminer les paramètres du graphe (nmax, n et m) et initialiser le tableau d'inividu. Et une deuxième pour remplir les listes de voisins.

– A chaque individu est associé un numéro d'individu et son numéro réel dans le fichier, ces données étant stockées dans la liste chaînée lStanford.

– À la fin du chargement on affiche une ligne pour indiquer les paramètres du graphe : nmax, n (nombre effectif de sommets), m (nombre d'arêtes) et degré moyen.

Modèle de prairie aléatoire:

- Nous avons choisi le modèle plus général Random Disk Graph pour chaque individu. Chaque individu se voit désigner une position aléatoire parmi celle encore disponible dans la prairie. De plus le rayon de chaque individu est déterminé aléatoirement entre deux valeurs qui sont rmin et rmax.
- Afin de modéliser l'infection du Cordyceps fungus sur la fourmis, nous avons besoin que le rayon des fourmis mortes augmente considérablement après la mort (pour modéliser la dissémination des spores du champignon).

Modèle de Watts-Strogatz:

Implémenté. Vérification après rebranchement que le nombre d'arêtes reste m=k*n.

Modèle de Barabási-Albert:

Implémenté.