Groupe Calculs de Cycles de protection pour réseaux
Léo Andéol
Lothair Kizardjian
Cyril Govin
Guillaume Chafiol

Analyse de l'existant

I – Introduction

Nous allons développer une application en C++, afin de calculer des cycles de protection pour réseaux. Pour cela nous calculerons des routes, ainsi que des cycles afin d'avoir des voies de secours pour les données en cas de panne. Nous confronterons ensuite plusieurs méthodes et plusieurs algorithmes pour voir lesquels sont les plus adaptés dans quels cas.
Mais premièrement, informons-nous sur l'éxistant : qu'est-ce qui a été fait concernant ce sujet, mais plus généralement, concernant les graphes en informatique, qui aurait un rapport avec notre projet ?

II – Information

II – 1 – Bibliothèques

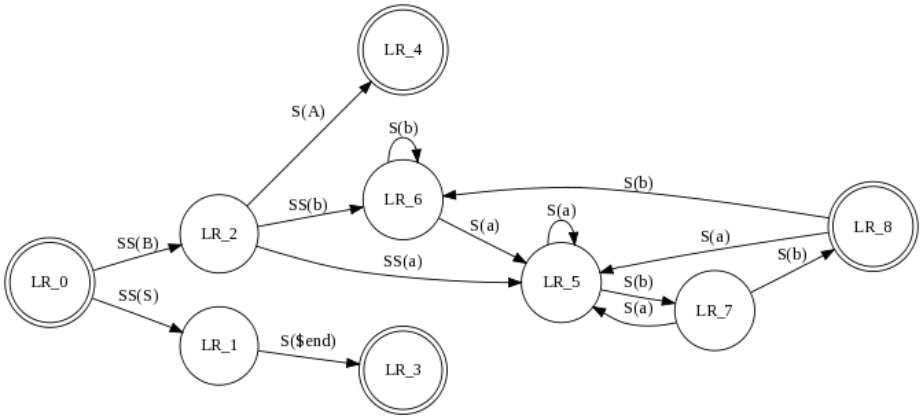
Il existe beaucoup de bibliothèques gérant les graphes, reparties sur plusieurs langages différents :

- C++ : LEDA (propriétaire), BGL (open source)
- C : iGraph (open source)
- R : iGraph (open source)
- Python : iGraph (open source), NetworkX (open source)
- Java : Jgraph (open source)

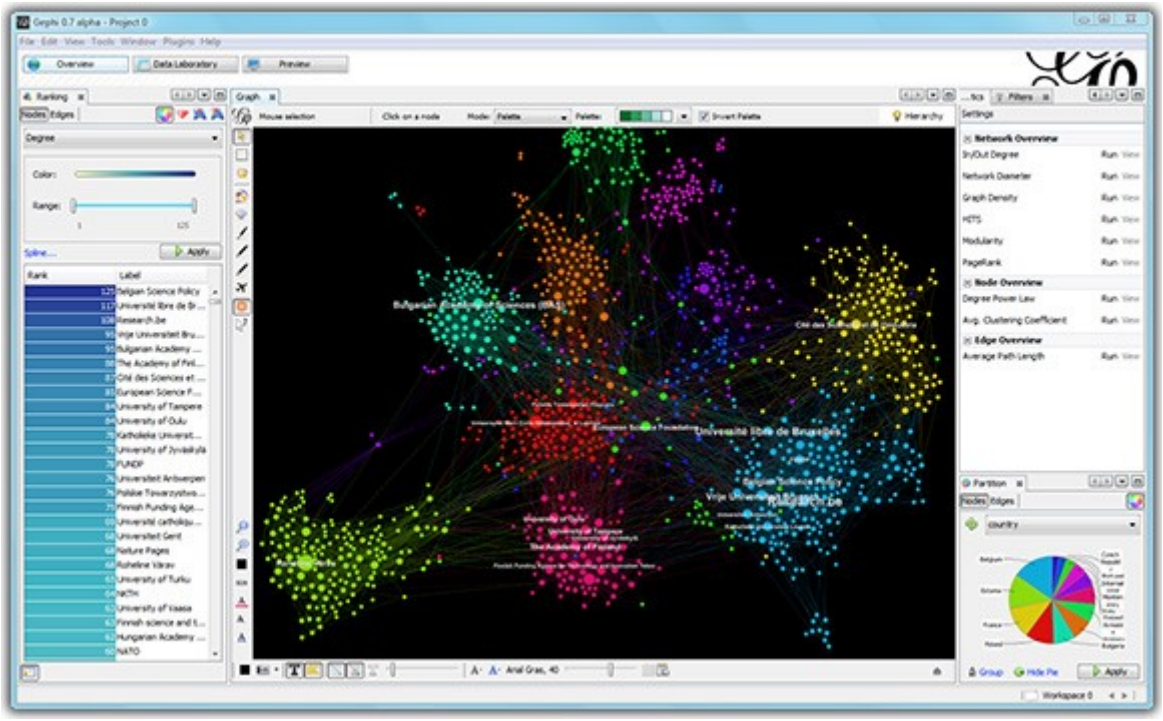
Il en existe en effet beaucoup plus que cela, mais nous nous limitons aux plus grandes et plus connues.

II – 2 - Visualisation

Etonamment, il semble que peu de logiciels de visualisation existent, graphviz est le principal, qui peut lire plusieurs types de graphes, et les exporter en tant qu'images ou autre, est open source, mais ne permet pas la création de graphes, et ne dispoce pas d'interface graphique, il permet juste la convertion de graphes en images.



D’autre part, le logiciel Gephi s’impose aussi, il est très moderne, puissant, permet la création de graphes et agréable à l’utilisation mais cependant il n’est pas open source et ne permet de n’utiliser que son format.



II – 3 – Algorithmes

Il existe énormément d’algorithme de calculs sur des graphes, qui ont des buts, et des façons de calculer tout à fait différentes, les trois grandes catégories sont :

- Algorithme de parcours en largeur : calcule étape par étape (les noeuds proches, puis proches des proches, etc ...) la distance de tous les noeuds par rapport à un noeud dit source.
- Algorithme de parcours en profondeur : trouve le chemin entre deux noeuds, en éxaminant chaque chemin possible, en allant au bout de chacun un part un.
- Algorithme de parcous à coût uniforme : trouve le chemin entre deux noeuds, étape par étape, comme l’algorithme de parcours en largeur, mais prend en compte la pondération des liens. Le but des algorithmes diffère aussi.

Au sein de ses catégories se trouvent beaucoup d’algorithmes, en voici quelques exemples :

- Algorithme du plus court chemin de Dijkstra
- Algorithme du plus court chemin de Bellman-Ford
- Algorithme du plus court chemin A*
- Algorithme des paires de Johnson
- Algorithme de l’arbre recouvrant minimum de Kruskal
- Algorithme de l’arbre recouvrant minimum de Prim
- Algorithme des composants connectés
- Algorithme des composants solidement connectés
- Algorithme des composants dynamiquement connectés

III – Analyse

III – 1 – Bibliothèques

Nous devons donc choisir une bibliothèque parmis toutes celles disponibles, et un langage.
Par rapport à notre cahier des charges, nous devons choisir un langage plus rapide afin de permettre de gérer des graphes larges, tout en restant dans des temps aussi courts que possible (une machine ne peut pas se permetre de prendre 1h+ pour calculer la meilleure route alternative en cas d'accident). Nous pouvons donc éliminer déjà les langages Java, Python et R car interprétés.
Ensuite, il serait préférable, en accord avec la mentalité GNU/Linux d'utiliser une bibliothèque open source, de préférence respectant un maximum le standard POSIX.
De plus il faudrait que la bibliothèque soit portable, et permette d'obtenir un visuel sur les graphes facilement.
Suivant ces conditions, le choix de la bibliothèque BGL (Boost Graph Library) émerge naturellement. Cette bibliothèque est open source, est très complète et permet de créer des graphes qui correspondent à notre besoin, de produire des fichiers DOT qui se convertissent en images grâce à graphviz, suis le standard POSIX, et puis surtout, boost est la principale bibliothèque en C++, après la bibliothèque standard, et est donc portable. Elle est immense et propose de nombreuses fonctionalités, comme la gestion des graphes, de polygones, de chronomètres, et bien plus. Cette bibliothèque inspire souvent le standard C++, par exemple avec la gestion des unordered_map en C++1.

III – 2 - Visualisation

En accord avec le choix de la bibliothèque boost, qui peut produire des fichiers DOT, nous utiliserons donc le programme graphviz qui permet de de transformer ces dits fichiers, en images.
Ensuite au cours de l'évolution du projet, nous développerons progressivement une GUI afin de limiter les dépendances, et augmenter l'ergonomie de notre programme.

III – 3 – Algorithmes

Le choix d'algorithme est plus compliqué, en effet l'algorithme devra être choisi en fonction de ce que l'on veut faire, par exemple les algorithmes de Dijkstra et de Kruskal ne donnent pas lieu à une comparaison, car ne calculent pas les mêmes informations.
De plus, suivant les graphes sur lesquels on devra appliquer l'algorithme choisi, l'éfficacité peut varier grandement; en effet nous pouvons prendre l'exemple des algorithmes de Prim et de Kruskal, comparons leur compléxité

- Prim : O(E + V log V)
- Kruskal : O(E log V)

E : liens
V : noeuds

Comme l'on peut le voir ci-dessus, en fonction du nombre de noeuds, et de liens, soit l'un soit l'autre sera plus éfficace. Ne connaissant pas en avance les détails des graphes que nous étudierons, et ayant à calculer plusieurs informations, nous ne choisierons pas d'algorithmes, et en

implémenterons un maximum, en comporant leur complexité afin d'adapter automatiquement notre choix d'algorithme au graphe étudié.

IV – Conclusion

Pour conclure, résumons nos choix quant à l'évolution de notre programme :

- Ecrit en C++, en utilisant la bibliothèque BGL
- Export de fichiers DOT visualisés par graphviz dans un premier temps, puis développement d'une interface graphique pour notre programme
- Implémentation de plusieurs algorithmes par catégories afin d'optimiser le temps de calcul en fonction du type de graphes que nous avons à étudier.