Compte rendu de la réunion du 6 Février

Algorithme de parcours

- Flyod-Warshall temporel
- Paramètres : Le graphe temporel et une matrice initialisée à $+\infty$ avec 0 sur la diagonale
- Complexité : Temps - $O(n^3 * t)$ Mémoire - $O(n^2 * t)$ avec n le nombre de liens
- Principe : 3 boucles impliquées pour calculer les distances de tout le monde à tout le monde, et mettre à jour les distances minimales si trouvées.
- Spécificités temporelles : 4ème boucle pour le temps, et si on dépasse T_{max} (Par exemple traverser un lien de poids 4 au temps 98/100), alors mettre la valeur à $+\infty$.
  De plus on compare à (dist[i,j][t+1] + 1) pour chaque temps t.
- Pourquoi celui-ci ? Algorithme facile à implémenter et adapter et est celui déjà utilisé partout pour les graphes non-temporels.
- Si trop coûteux, on approximera plusieurs pas de temps en un seul.
- Autres algorithmes à creuser si possible : Johnson, Dijkstra

La mesure demandée

- Paramètres : T_{max}, calculé avec le diamètre de notre graphe. Intervalle δ, qui sera notre fréquence de mise à jour des attaques des temps 0 à T_{max}.
- Si division coûteuse, on peut trouver d'autres formules que l'efficacité évaluer la robustesse du graphe tant que notre valeur reste cohérente et interprétable.
- Pour pouvoir la calculée, nous sommes parti sur une implémentation de Time-varying graph. Mais il reste à régler le problème des pas de temps intermédiaires :
  - Si nous faisant des pas de temps de $2x$, comment faire si nous avons un lien qui coûte $3x$ ou $1x$?
  - Cela implique une décorrélation entre les temps de l'évolution du graphe et les poids de celui-ci.
  - Pour l'instant : on ne prend pas en compte les changements de rue pendant qu'on la traverse. A réfléchir pour mieux résoudre le problème, comme sous-diviser nos pas de temps par exemple.

Attaques

- Budget par nombre de voies (Coût pour bloquer une rue relative à son nombre de voies).
- A faire : mêmes attaques mais sans ce coup par nombre de voies d'une rue.
- Faire attention pour l'attaque se déplaçant à ne pas avoir deux bloque-liens superposés.
- Pour les attaques statiques, on réimplémente des Graph Equivalent Stream (à voir pour optimiser les calculs spécialement pour ça)
- A creuser : Les attaques en coupe de graphe (Séparer le graphe en 2 composantes connexes ou plus) (cf. L'algorithme d'Alexis)
  Avec un algorithme propre ou avec les fonctions de networkx (Avec l'option strongly connected pour les graphes orientés)

Données des villes

- Pour les données manquantes sur OSMnx : On a choisi 30km/h pour la vitesse par défaut et un nombre de voies à 1.
  Il faut extrapoler et donc choisir des données par défaut cohérentes.
- Pour calculer le poids des liens, tester si *distance* ou $\frac{distance}{vitesse}$ est le plus pertinent.
- Note : quand il n'y a pas d'intérêt évident de choisir quelque chose lors d'une démarche scientifique, il faut tenter les 2 approches et voir laquelle est la plus pertinente.

Autres

- Pour tester nos algorithmes : Prendre des villes "typiques" pour avoir le cas le plus général possible.
- S'il nous reste du temps après, on peut aussi travailler sur des graphes de routes maritimes et calculer les impacts de grèves de ports ou d'accidents de bateau pour la circulation.
- Pour optimiser les exécutions en parallèle sur les serveurs du lip6, exécution par stratégies et par pas de temps sur la même ville pour pouvoir partager son graphe en mémoire et éventuellement d'autres données