RENIN : Analyse de réseaux avec Python

Chaque exercice doit faire l'objet d'une sauvegarde dont le nom comportera le numéro de l'exercice : « exo1a.py », puis « exo1b.py », puis « exo1c.py »… **Ces fichiers sont à envoyer sous la forme d'un dossier zippé à l'adresse mail suivante : serge.lhomme@u-pec.fr**

Appui sur l'existant

1) Diversification des stratégies d'attaque et amélioration du code « vul2.py »

a) Dans le fichier « vul2.py », ajoutez un scenario qui s'appuie sur un indicateur de proximité : nx.closeness_centrality(G). Ce scenario doit perturber les sommets dont les valeurs de proximité sont les plus fortes (en l'occurrence, à la fin des itérations, ce sont les 10 sommets qui ont les valeurs les plus fortes qui seront perturbés).

Si vous observez tous les scénarios produits, il existe des séquences de code communes. Pour rendre notre code plus élégant et pour éviter ces répétitions, il est possible de créer des fonctions. Ci-dessous, vous trouverez un exemple de fonction. « calculbidon » correspond au nom de la fonction, « a » est une donnée d'entrée, « y = 2 + a » et « print(y) » sont les opérations à effectuer. La dernière ligne sert à exécuter cette fonction pour une valeur particulière. La commande « return » permet par la suite de récupérer une valeur au sein d'une variable (en l'occurrence la variable z).

```
def calculbidon(a):
    y = 2 + a
    print(y)
    return y
z=calculbidon(3)
calculbidon(2)
```

b) En vous inspirant de l'exemple donné, placez (juste après les importations de bibliothèques) une fonction « perturbation » qui prendra en entrée l'identifiant d'un nœud et le graphe à perturber. Cette fonction attribuera un poids de 1000000 aux arcs adjacents à ce nœud afin de simuler sa perturbation.

c) De même, créez une fonction « eval_perturb » qui permettra d'évaluer le nombre de relations impossibles au sein d'un graphe donné. En entrée, cette fonction requiert uniquement le nom du graphe. Elle doit impérativement retourner une valeur «valc».

d) Placez ces fonctions au sein d'une stratégie d'attaque.

Partie exploratoire

Vous allez chercher à identifier les combinaisons de sommets dont les suppressions seraient les plus problématiques. Pour cela, vous n'allez pas vous appuyer sur des stratégies, mais sur des algorithmes d'optimisation.

2) Simuler toutes les combinaisons possibles impliquant deux sommets

a) En repartant d'un nouveau fichier, importez les bibliothèques numpy, scipy et networkx, puis ajoutez le tableau « arcs_w ».

b) Ajoutez les fonctions « perturbation » et « eval_perturb » développées précédemment.

c) Importez une nouvelle bibliothèque : « import itertools as iter ». En effet, Cette bibliothèque permet d'appeler une commande listant toutes les combinaisons possibles de deux sommets : list(iter.combinations(range(57), 2))

d) Créez une boucle qui permettra de parcourir le tableau produit par la commande ci-dessus et affichez l'ensemble de ces combinaisons une par une.

e) Profitez de cette boucle et des fonctions développées pour évaluer les impacts générés par tous les scénarios impliquant la perturbation de deux sommets. N'oubliez pas de créer à chaque itération un nouveau graphe valué nommé G à partir du tableau « arcs_w ». Affichez les résultats correspondants.

f) Identifiez la combinaison de deux sommets dont la perturbation produirait les plus forts impacts. Pour cela, stockez tous les résultats dans un tableau, puis triez ce tableau.

3) Programmer un algorithme glouton

Un algorithme glouton est un algorithme qui, étape par étape, retient la solution qui lui paraît la plus optimale sans jamais remettre en question ses anciennes décisions. Vous avez identifié la combinaison de deux sommets (39 et 51) dont la perturbation produirait les plus forts impacts. On va s'appuyer sur ce résultat pour chercher ce qui pourrait être la combinaison de trois sommets dont la perturbation produirait les plus forts impacts.

a) Utilisez le résultat de 2f afin de perturber un graphe nommé G2.

b) Créez une boucle qui va parcourir l'ensemble des sommets du graphe G2 pour étudier un nombre limité de combinaisons possibles de trois nœuds (en l'occurrence seulement 57 combinaisons).

c) Identifiez dès lors le scénario le plus problématique parmi ces scénarios de trois nœuds perturbés.

4) Programmer un algorithme de voisinage

Un algorithme de voisinage va s'appuyer sur une solution existante et va tenter de l'améliorer. Dans les faits, un algorithme glouton identifiera le sommet 8 comme le plus problématique, puis lui associera le sommet 53. Il aboutira pour trois nœuds à une égalité entre un grand nombre de scénarios. L'algorithme de voisinage va nous permettre d'améliorer cette solution.

a) On considère que la solution retenue par l'algorithme glouton correspond à la perturbation des nœuds 8, 53 et 55. Créez une boucle qui va remettre en cause le choix du premier sommet de ce trio. Conservez le scénario le plus problématique.

b) Créez une double boucle qui permettra de remettre en question tour à tour tous les sommets du trio 8, 53 et 55. A la fin de chaque remise en question, repartez avec la nouvelle solution optimale identifiée. Conservez le scénario le plus problématique.