TP Algorithmique MPRI

Marc-Antoine Weisser

28 septembre 2023

Résumé

L'objectif de ce TP est de mettre en oeuvre la démarche présentée dans ce cours. Dans un premier temps, nous nous intéresserons à la modélisation du problème à partir d'un cahier des charges. Ensuite nous aborderons les questions de la complexité et d'approximabilité. Nous implémenterons au moins deux méthodes de résolutions (approximation, branch and bound, recuit, ...). Enfin nous interrogerons sur les avantages et inconvénients de avec une évaluation de performance.

Ce TP est mis en scène, prenez soin d'analyser ce que les interlocuteurs disent. Les dialogues contiennent des informations importantes.

1 Travail attendu

Le TP se déroule en 3 séances et pourra être terminé à la maison. Il peut être réalisé en groupe de deux étudiants, mais attention, le travail attendu est un peu plus conséquent dans ce cas (voir plan du rapport).

Le TP donnera lieu à un rapport rédigé en Latex. Il devra être structuré selon le plan présenté plus loin. Nous acceptons des rapports en français ou en anglais. Le code source de vos programmes devra être commenté et fourni en même temps que le rapport. L'ensemble de votre travail doit être regroupé dans une archive zip ou tar.gz (nous n'acceptons pas de rar, de bzip, ou autre 7zip). Cette archive est organisée de la manière suivante. Un dossier parent intitulé `rapport_algo_NOM` (ou `rapport_algo_NOM1_NOM2` si vous êtes en groupe) contiendra le PDF du rapport ainsi qu'un sous-répertoire "source" contenant le code de vos programmes ainsi que les données.

Voici le plan du rapport. Vous le compléterez avec les informations que vous obtiendrez au fils des séances de TP. Si vous travaillez seul, vous pourrez choisir de ne pas implémenter la méthode exacte.

- Modélisation
  - Présentation du problème
  - Problème de décision
  - Problème d'optimisation
- Résultats Théoriques
  - Complexité (preuve de NP-complétude)
  - Approximabilité
    - Principaux résultats connus et référence vers un compendium
    - Présentation d'un algorithme d'approximabilité
    - Preuve du rapport d'approximation de l'algorithme présenté
- Méta-heuristique (Recuit)
  - Description de la méthode
  - Voisinage
- Méthode exacte (Branch & Bound) [option si seul, obligatoire en groupe]
  - Description de la méthode
  - Présentation de la borne
- Évaluation de performance

Durant les séances de TP, il est vivement recommandé d'échanger avec les autres groupes d'étudiants : ne restez pas seul.

Vos encadrants vous contacterons pour fixer un moment d'échange, entre chaque séance, en visio. Cela permettra échanger, de connaitre votre avancement et de vous débloquer le cas échéant.

2 Modélisation

2.1 Cahier des charges

Vous êtes Camille et vous travaillez depuis peu dans un laboratoire de recherche en informatique. Vous êtes contacté par Alix qui travaille dans une société d'installation de fibre optique : Aerian Fiber. Son entreprise est confronté à un problème d'optimisation et n'arrive pas à trouver de solution satisfaisante. Voici une retranscription de votre premier rendez-vous.

ALIX: Il y a déjà quelques années, l'état a lancé un plan de financement publique pour l'installation de la fibre optique. Ces subventions sont essentiellement destinées au déploiement de réseau optique dans les zones "non conventionnées". Ce sont des zones où aucun opérateur n'a déjà déployé de réseau optique, notamment parce que le rapport investissement/bénéfice n'est pas assez intéressant. En campagne principalement. Notre société répond à ce type de demande. Notre spécificité c'est d'utiliser les infrastructure de télécom existante et d'y accrocher des fibres.

CAMILLE: Accrocher ? Dans le réseau aérien ?

ALIX: Oui, tout à fait. Pour un déploiement de réseau souterrain, il est nécessaire de faire passer les fibres dans des gaines déjà enfouies ou même de creuser pour installer les gaines. C'est très couteux et les collectivités territoriales qui nous financent n'ont que rarement les moyens. Nous préférons donc utiliser les poteaux existants le long des routes. Il y en a déjà quasiment partout, pour porter les installations électriques ou téléphoniques. On peut donc y ajouter une fibre optique. Ça demande parfois de changer le poteau mais c'est bien moins cher que l'enfouissement.

CAMILLE: Ok, je vois l'idée mais en quoi notre laboratoire peut être utile ?

ALIX: Pour répondre au appel à projet, nous avons besoin d'être concurrentiel. Cela veut dire, en particulier, proposez le réseau nécessitant le moins d'installation possible.

CAMILLE: Qu'est ce que vous entendez par le moins d'installation possible ?

ALIX: Et bien, nous avons un à notre disposition la carte des installations électrique et télécom. Ce sont essentiellement des poteaux qui suivent des routes et les intersections. Le coup d'installation d'une fibre le long d'une route est connue, globalement cela est proportionnel à sa longueur.

CAMILLE: Ok, c'est pour nous ce que l'on appelle un graphe. Vos intersections sont les noeuds et les routes sont les arêtes. On peut associer un poids à chaque arrête pour représenter le cout d'installation.

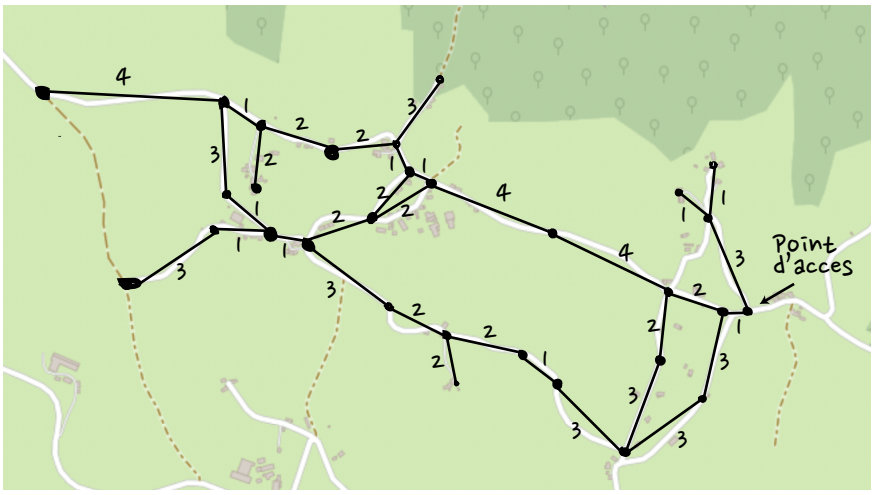
ALIX: Oui, ça me semble correspondre.

CAMILLE: Et moi ça me semble relativement simple alors. Vous cherchez à créer une structure qui connecte toutes les intersections de votre graphe. Pour que cette structure soit de poids minimum, il faut que ce soit un arbre. Vous cherchez ce que l'on appel l'arbre couvrant de poids minimum.

ALIX: De poids minimum ?

CAMILLE: Oui, le poids d'un arbre est le poids de la somme des arrêtes qui le constitue. On cherche donc, parmi tous les arbres, celui qui a un poids le plus petit. Celui qui nécessitera le moins d'investissement pour l'installation de la fibre.

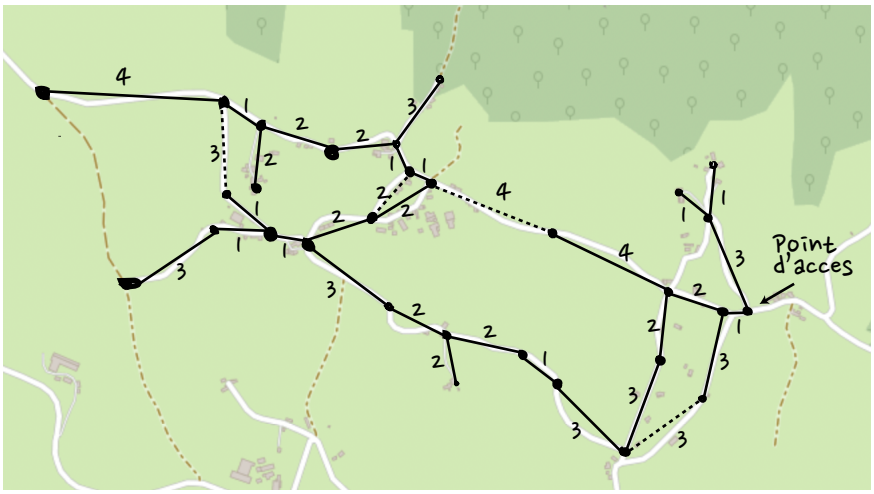
ALIX: Attendez, on va faire un exemple, ça sera plus clair. Je dois avoir un exemple de topologie. Un réseau avec une estimation du prix d'installation de la fibre le long des axes.



CAMILLE: Vous avez marqué "point d'accès", c'est le point d'inter-connexion avec le réseau national ?

ALIX: Exactement. Le point d'accès doit donc alimenter tous les lotissements. Mais les liens fibres sont bidirectionnels alors le rôle du point d'échange n'est pas particulier. Nous parlons de terminaux dans notre jargon. Un terminal peut être un lotissement ou le point d'échange. Ce qui est important pour nous c'est qu'il y ait un chemin entre tout couple de terminaux.

CAMILLE: Ok, parfait donc je peux calculer et représenter l'arbre couvrant de poids minimum en trait plein. Les pointillés sont des liens dont on a pas besoin. Cet arbre on le trouve facilement avec l'algorithme de Kruskal ou de Prim. Il couvre tous les sommets et est de poids minimum. C'est un algorithme dont la complexité est polynomial, donc il faudrait vraiment des très grosses topologie avant que cela pose un problème.

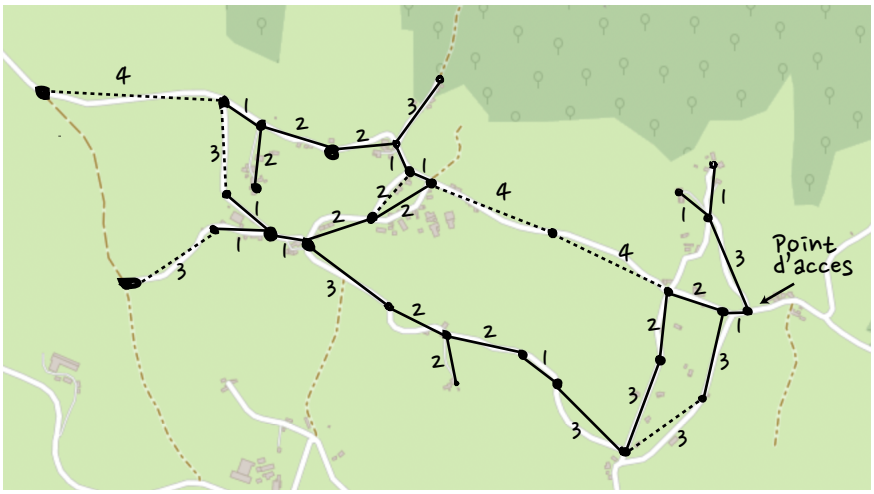


ALIX: J'ai un doute, que voulez vous dire par "couvrir tous les sommets" ?

CAMILLE: Et bien tous les nœuds du graphe appartiennent à l'arbre, ils sont couverts.

ALIX: D'accord, mais pourquoi couvrir ces intersections. Ça ne sert à rien, il n'y a pas de lotissement ici ! ?

CAMILLE: C'est pas faux... Il faudrait quelque chose qui couvre uniquement les terminaux.



ALIX: Après ce n'est peut être pas un problème ?

CAMILLE: Et bien je ne sais pas. Enfin, ce n'est pas un soucis, c'est le problème que vous cherchez à résoudre. Mais il faut que je creuse. Ce que je vous propose c'est de regarder comment modéliser le problème et revenir vers vous pour qu'on puisse valider cette piste.

ALIX: Parfait. Faisons comme ça.

2.2 Problème d'optimisation

Nous voici prêt à nous lancer dans cette histoire. Suite à la première réunion avec Aerian Fiber, vous vous mettez au travail. Il convient dans un premier temps de formaliser le problème d'optimisation. Le rédiger le sous une forme normalisée :

— quelles sont les données d'entrées ?
— qu'est ce qu'une solution réalisable ?
— quel est le poids d'une solution ?

Vous vous rappelez qu'il existe des listes de problèmes d'optimisation classiques et qu'il est sans doute une bonne idée de commencer par regarder si un problème correspondrait. C'est sans doute un problème de réseau et en particulier un problème d'arbres. On trouve ce type de problème dans le compendium de Viggo [1].

Question 1. *Recherchez dans le compendium le problème qui correspond selon vous.*

Indice. *Pour savoir si vous avez trouvé le bon problème, copiez l'adresse web de la page décrivant le problème. Elle devrait être de la forme :* `https://www.csc.kth.se/~viggo/wwwcompendium/nodeXX.html` *ou XX est un nombre. Calculez le hash de cette adresse en utilisant ce site* `https://www.sha1.fr/`*. Si vous avez trouvé le bon problème, le hash de l'adresse devrait être*

$$0342e07ad89553a6052357b732221f69e58bd2f9$$

Question 2. *Intégrez dans votre rapport le problème que vous avez trouvé dans le compendium en conservant le formalisme.*

2.3 NP-complétude

Bravo, vous avez trouvé un problème qui semble correspondre au problème que vous a décrit Alix, votre interlocuteur chez Aerian Fiber. Vous décidez de confronter votre idée auprès de Charlie qui travaille avec vous et connait bien ces sujets.

CAMILLE: Voici ce que j'ai reussi a identifier en utilisant le compendium.
CHARLIE: C'est pas mal. Je pense que tu as raison mais je suis pas sur que ce soit une bonne nouvelle.
CAMILLE: Comment ça ?
CHARLIE: Tu es au courant que tous les problèmes de cette liste sont NP-complet ? Pour aucun d'entre eux il n'existe d'algorithme de résolution en temps polynomial.
CAMILLE: A priori...
CHARLIE: Certe, a priori... Mais sans te manquer de respect, je ne crois pas que ce soit toi qui prouvera que P = NP.
CAMILLE: Ok, merci, rappelles-moi de plus rien te demander !
CHARLIE: Bah écoutes, je me trompe peut être sur ton compte !
CAMILLE: Tout à fait.
CHARLIE: Alors j'ai un petit challenge pour toi. Je pense savoir comment on démontre que ce problème est NP-Complet. Ça devrait pas être dur pour toi ?
CAMILLE: Challenge accepté ! Donc déjà on parle du problème de décision associé. Ça n'a pas de sens de parler de NP-complétude pour un problème d'optimisation.

1. `https://www.csc.kth.se/tcs/compendium/`

CHARLIE: Ok, c'est vrai, c'est un abus de langage. Mais t'as raison, on parle du problème de décision. On ne cherche donc plus un arbre de poids minimal mais un arbre dont le poids est inférieur à un paramètre K.

CAMILLE: Ça me va. Et clairement le problème est dans NP.

CHARLIE: Oui, c'est sur. Si on a dispose d'une instance positive et d'un arbre solution, on peut vérifier en temps polynomial que cet arbre vérifie les contraintes et la limite de poids K.

CAMILLE: Bien. Je suppose que tu veux en plus une réduction polynomiale ?

CHARLIE: Évidemment... Mais je veux bien te donner un indice ! C'est une réduction qui va de Exact Set Cover vers notre problème.

CAMILLE: Bon... tu me laisse 5 minutes ?

CHARLIE: Écoutes, j'ai une réunion qui débute, envoie moi ta proposition par mail. Je devrais avec un moment pour la lire, mon aprèm ne s'annonce pas passionnant...

Vous repartez à votre bureau avec la ferme intention de démontrer à Charlie que vous avez le niveau. Vous ouvrez un navigateur et commencez par trouver une définition d'Exact Set Cover [2]. Vous vous mettez alors à la recherche d'une réduction qui conviendrait.

Question 3. *Trouvez une réduction polynomiale qui pourrait convenir. Vous pouvez la découvrir par vous même ou bien la trouver sur Internet mais dans ce second cas, il convient de vous la réapproprier. Dans tous les cas, intégrez la preuve de NP-complétude (appartenance à NP et réduction) à votre rapport.*

2.4 Résultats Théoriques

Votre réduction polynomiale terminée, vous l'envoyez à Charlie. Vous n'attendez pas sa réponse pour relire les résultats théoriques présentés dans le compendium et pour passer un coup de fils à Alix. Vous discutez rapidement de votre modèle pour valider que cela colle bien. Plus tard dans l'après midi, vous vous croisez à nouveau Charlie.

CAMILLE: Alors cette réduction ? T'en penses quoi ?

CHARLIE: Et bah rien du tout, je l'ai pas lu !

CAMILLE: ...

CHARLIE: Par contre, j'ai quand même bossé pour toi. J'ai regardé rapidement les résultats connus pour ce problème et j'ai même exhumé un algorithme approché que j'ai retrouvé dans un vieux cours.

CAMILLE: Ah top ! Alors ?

CHARLIE: Déjà, il y a une bonne nouvelle. Il existe un algorithme d'approximation.

CAMILLE: Un algorithme d'approximation ?

CHARLIE: Un algorithme qui s'exécute en temps polynomial et renvoie une bonne solution.

CAMILLE: Une solution exacte ?

CHARLIE: Ah non, le problème est NP-complet, donc à moins que P=NP il n'existe pas d'algorithme polynomial pour résoudre le problème de manière exacte.

CAMILLE: Alors c'est quoi une bonne solution ?

CHARLIE: C'est une solution qui est à une certaine distance de l'optimal. Dans notre cas le meilleurs algorithme d'approximation à rapport constant que l'on connaisse fournit des solutions qui sont au pire 1.55 fois plus grande que l'optimum. Pour être exact c'est $1+\ln(3)/2$ mais je chipote.

2. `https://en.wikipedia.org/wiki/Exact_cover`

CAMILLE: Ok, donc la distance à la solution optimale pour un algorithme approché est une constante multiplicative ?

CHARLIE: Oui, dans le cas des problèmes de minimisation c'est une borne supérieur entre la solution renvoyée par l'algorithme d'approximation et la solution exacte. C'est une valeur qui vaut toujours plus de 1.

CAMILLE: Et pour les problème de maximisation ?

CHARLIE: Dans ce cas, le rapport est toujours en 0 et 1.

CAMILLE: Et le rapport d'approximation est toujours une constante multiplicative ?

CHARLIE: Ce n'est pas toujours une constante. Ça peut être une fonction dépendant de la taille de l'entrée du problème. Par contre j'ai une mauvaise nouvelle. De ce que j'ai lu, le problème est APX-complet. Cela veut dire qu'il n'existe pas d'approximation dont le rapport peut être aussi proche de 1 de l'on veut. C'est ce qu'on appelle les schémas d'approximation. Un algorithme de rapport d'approximation $1+\epsilon$ dont la complexité est en $\mathcal{O}(n^{\frac{1}{\epsilon}})$ voir même parfois polynomiale [3] en n et en $\frac{1}{\epsilon}$. Ce n'est pas notre sujet puisque, dans notre cas, il n'y en a pas. Laissons ça pour une autre fois.

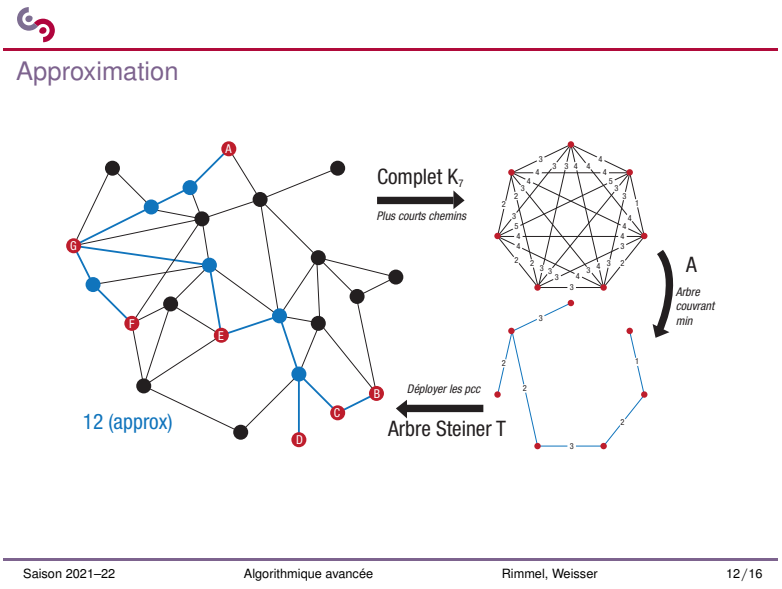
CAMILLE: Ça me va. Du coup, tu m'as parlé d'un cours que tu as retrouvé. Ça parlait de quoi ?

CHARLIE: Et bien justement d'un algorithme d'approximation pour ton problème. Ce n'est pas le meilleurs rapport, il vaut seulement 2. Par contre, il est simple à coder, tu devrais pouvoir l'implémenter rapidement. Ça te permettra de faire une petite démo la prochaine fois que tu iras à Aerian Fiber.

CAMILLE: Top ! Envoie moi tes slides par mail, je regarde ça demain.

CHARLIE: Yep. Une dernière chose, tu verras que dans mes slides, l'algo est présenté pour un graphe non pondéré mais ça fonctionne aussi si les arêtes sont pondérées. Et puis quand t'auras fini, on parlera d'évaluation de performance. Parce qu'il y a quelque chose à regarder de ce coté là aussi...

Question 4. *Rédigez l'algorithme de 2-approximation et sa preuve. Intégrez le tout à votre rapport.*



3. $\mathcal{O}(n^c \left(\frac{1}{\epsilon}\right)^d)$ avec c et d deux constantes

Preuve de l'approximation

Preuve

- $T^* \leq T \leq A$
- On veut montrer que $A \leq 2T^*$
- Soit C un cycle eulérien obtenu en duplicant les arêtes de T^*, $C = 2T^*$
- On parcours les terminaux de C par ordre de première apparition et on construit C' en utilisant des pcc $C' \leq C$
- C' correspond à un cycle dans K, notons le C''
- On supprime une arête à C'', on obtient un arbre A''
- $A'' \leq C'' = C' \leq C = 2T^*$
- Tout arbre couvrant minimum de K a un poids $A^* \leq A''$
- $T^* \leq A^* \leq A'' \leq C'' = C' \leq C = 2T^*$

2.5 2-approximation

Votre première journée de travaille a été consacré aux résultats théoriques et vous avez hâte de débuter la deuxième pour passer à la mise en œuvre. En arrivant au laboratoire nous passer voir Charlie pour recueillir ses précieux conseils avant de vous lancer.

CHARLIE: Alors cet algo ? Tu l'as bien compris ?

CAMILLE: Oui, je pense. Il n'est pas très compliqué. Maintenant je m'attaque au développement.

CHARLIE: Tu vas utiliser quel langage et quelle bibliothèque ?

CAMILLE: Je ne sais pas encore. Je me dis que si je veux que ça aille vite il vaut mieux que je le code en C.

CHARLIE: ah ouais ? Mais qu'est ce qui est le mieux ? Que tu développes rapidement ou que le programme s'exécute rapidement ? Parce que pour une démo, autant que tu aille vite. Rien dans ton algorithme n'est très couteux, donc pas besoin de viser l'optimisation du code a tout pris. Si j'étais toi, je coderais ça en Python avec la bibliothèque NetworkX. Elle intègre tous les algos classiques de graphe, tous ceux dont tu as besoin. Mais fait comme tu veux.

CAMILLE: Autant que ce soit la machine qui perde du temps plutôt que moi...

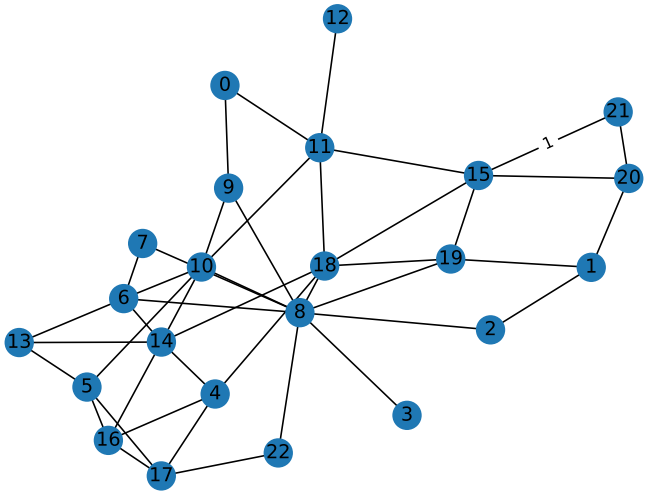
CHARLIE: C'est l'idée... Je vais te montrer rapidement. On va commencer par les manipulations de base. Par exemple, voici comment importer le module networkx et créer une fonction qui fabrique le graphe présentant la 2-approximation. Tu vois qu'on peut créer un graphe vide et y ajouter les sommets, les uns après les autres avec la méthode `add_node(i)` qui prend en paramètre l'étiquette du sommet ; ou plusieurs d'un coup avec `add_nodes_from(seq)` qui prend en paramètre une séquence d'étiquettes. C'est la même chose pour les arêtes. Ici elles n'ont pas de poids mais on peut tout à fait ajouter des pondérations. Tu peux trouver la documentation de ces fonctions en ligne [4].

4. `https://networkx.org/documentation/networkx-1.7/reference/classes.graph.html`

```
import networkx as nx

def create_test_graph():
    G = nx.Graph()
    G.add_node(0)
    G.add_nodes_from(range(1,23))
    G.add_edge(0, 9)
    G.add_edges_from([(0, 11), (1, 2), (1, 19), (1, 20), (1, 2),
    (2, 8), (3, 8), (4, 14), (4, 16), (4, 17), (4, 18), (5, 10),
    (5, 13), (5, 16), (5, 17), (6, 7), (6, 8), (6, 10), (6, 14),
    (6, 13), (7, 8), (8, 9), (8, 10), (9, 10), (10, 11), (10, 14),
    (11, 12), (11, 15), (11, 18), (13, 14), (14, 16), (14, 18),
    (15, 18), (15, 19), (15, 20), (15, 21), (16, 17), (17, 22),
    (18, 19), (18, 8), (19, 8), (20, 21), (22, 8)])
    return G
```

CHARLIE: On peut récupérer facilement la liste des sommets et d'arêtes du graphe. On peut fixer ou modifier le poids d'une arrête. Et surtout on peut afficher le graphe en utilisant le module `matplotlib` et le sauver sous forme de PDF. C'est assez pratique pour faire des tests.

```
def test():
    G = create_test_graph()
    print(G.nodes)
    print(G.edges)
    G[21][15]["weight"] = 1

    pos = nx.spring_layout(G)
    nx.draw(G, with_labels=True, pos=pos)
    labels = nx.get_edge_attributes(G, "weight")
    print(labels)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
    plt.savefig('test_graph.pdf')
    plt.show()
```



CHARLIE: Bon y a quand même quelques limites. On ne maitrise par exemple pas vraiment le placement des sommets. La bibliothèque essaie de les répartir au mieux mais ce n'est pas toujours lisible. En tout cas, voici notre graphe de test.

CAMILLE: Ça vaudrait le coup de vérifier que l'on a pas fait d'erreur en le saisissant mais peut importe. Ça fera l'affaire.

CHARLIE: Oui, effectivement ce n'est pas capitale. Tu ne te serviras sans doute pas beaucoup de ce graphe, à part pour quelques tests. À terme il faudra sans doute générer un grand nombre de graphes aléatoires. Ça aussi c'est relativement facile. La bibliothèque nous permet d'en générer suivant différents modèles. Par exemple, voilà comment faire un graphe aléatoire binomial.

CAMILLE: Comme la loi binomiale ?

CHARLIE: Oui. On choisit le nombre de sommets que l'on souhaite et une probabilité d'apparition d'une arrête entre deux sommets.

CAMILLE: Rien ne garanti que le graphe soit connexe ?

CHARLIE: Non. Mais on peut simplement tester si un graphe est connexe et re-générer un graphe le cas échéant.

```
def random_binomial_graph(n, p=0.5):
    G = nx.generators.binomial_graph(n, p)
    while nx.is_connected(G) is False:
        G = nx.generators.binomial_graph(n, p)
    return G
```

CAMILLE: À quoi correspond la valeur 0.5 qu'on associe à p dans la définition de la fonction ?

CHARLIE: C'est un paramètre optionnel. Si on ne donne qu'un seul paramètre à l'appel de la fonction, le second vaudra automatiquement 0.5.

CAMILLE: Pratique.

CHARLIE: Bien sur, tous les générateurs de `networkx` sont documenté [5]. Et il n'y a pas que des générateurs aléatoires. Par exemple tu as ici une méthode pour fabriquer un graphe complet : `nx.complete_graph`. La fonction prend en paramètre soit le nombre de sommets du graphe, soit une séquence contenant les étiquettes des sommets.

CAMILLE: La séquence des étiquettes ?

CHARLIE: Oui, par défaut les n sommets sont numérotés par des entiers de 0 à $n-1$ mais rien n'interdit de leur associer d'autres entiers ou même des lettres. Donc on peut passer par exemple la liste `["A", "B", "C"]` en paramètre. Le graphe complet aura alors 3 sommets étiquetés "A", "B" et "C".

CAMILLE: Si on revient à la génération aléatoire. Je vois que tu as utilisé une fonction toute faite pour tester si un graphe est connexe. C'est assez pratique. Est-ce qu'il en existe d'autres ? Je pense en particulier à la 2-approximation. Pour l'implémenter il me faut des plus courts chemins et arbre couvrant de poids minimum.

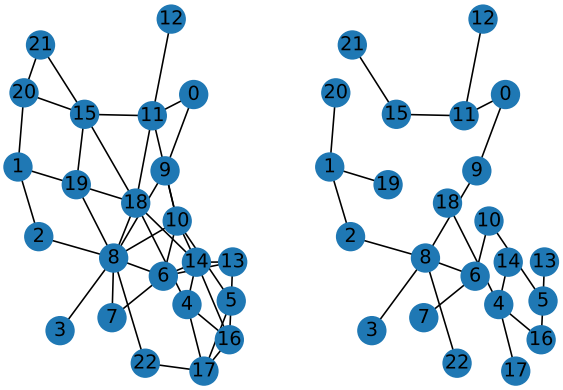
CHARLIE: Oui, tout à fait. Et c'est pour ça que `networkx` est si pratique. Par exemple, on ajoute des poids aléatoires sur toutes les arrêtes du graphe et calcule un arbre couvrant de poids minimum. On a une fonction dédiée. Elle est simple à intégrer et on peut la combiner avec les "subplot" de matplotlib.

```
def test_spanning_tree():
    G = create_test_graph()
    for u, v in G.edges:
        G[u][v]["weight"] = random.randint(1, 10)
    T = nx.algorithms.maximum_spanning_tree(G, weight="weight")
    print(T)

    plt.subplot(121)
    pos = nx.spring_layout(G)
    nx.draw(G, with_labels=True, pos=pos)

    plt.subplot(122)
    nx.draw(T, with_labels=True, pos=pos)
    plt.savefig('test_minimum_tree.pdf')
    plt.show()
```

CAMILLE: Alors effectivement, le calcul de l'arbre couvrant de poids minimum c'est presque gratuit. Par contre je comprends mal cette histoire de `subplot`. À quoi correspondent les valeur 121 et 122 ?

CHARLIE: On peut dire à `matplotlib` de dessiner plusieurs fois sur un même graphique. L'instruction `plt.subplot(121)` indique à `matplotlib` qu'il doit découper son espace en une ligne et deux colonnes. C'est le sens de la centaine et de la dizaine dans 121. Chaque morceau de l'espace est numéro en partant du haut à gauche. C'est ce que désigne l'unité. Donc 121 c'est la case de gauche et 122 la case de droite. Dans le doute, le mieux c'est de regarder la documentation de `matplotlib`. C'est assez complet et il y a plusieurs manières d'arriver au même résultat [6].

5. `https://networkx.org/documentation/stable/reference/generators.html`
6. `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html`



CAMILLE: Parfait, on a bien un arbre couvrant, 21 arêtes pour 22 sommets. Bon, évidement il faudrait vérifier les pondérations des arrêtes.

CHARLIE: Oui, faisons confiance à Python sur ce coup là. Allez, tant qu'on y est, regardons pour les plus courts chemins. Ça se fait bien également. On utilise la fonction `nx.algorithms.all_pairs_dijkstra_path`. Elle prend en paramètre un graphe et renvoie les plus courts chemins. Pour l'exemple on va afficher le chemin entre le sommet 0 et le 22.

```
def test_shortest_paths():
    G = create_test_graph()
    for u, v in G.edges:
        G[u][v]["weight"] = random.randint(1, 10)

    distances = dict(nx.algorithms.all_pairs_dijkstra_path(G, weight="weight"))

    pos = nx.spring_layout(G)
    nx.draw(G, with_labels=True, pos=pos)
    labels = nx.get_edge_attributes(G, "weight")
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
    plt.savefig('test_shortest_paths.pdf')
    plt.show()

    print(distances[0])
    print(distances[0][22])
```

CAMILLE: Ok, ça me semble assez clair. Sauf peut être la valeur renvoyer par la fonction `all_pairs_dijkstra_path`. Pourquoi y a-t-il besoin de la transformer en dictionnaire ?

CHARLIE: Ce que renvoie la fonction c'est un itérateur, un objet qu'on ne peut pas manipuler directement avec la notation crochet. Il faut le convertir en dictionnaire.

CAMILLE: Et la fonction ne renvoie que les plus courts chemins, pas leur poids ?

CHARLIE: Non, il faut le calculer ensuite si l'on en a besoin. Sinon on peut utiliser la fonction `all_pairs_dijkstra_path_length` mais elle ne renvoie que les distances et pas les chemins...

CAMILLE: C'est un peu galère ça, non ?

CHARLIE: Oui, j'avoue. Mais pour ce que tu as à faire, le travail est déjà bien mâché. Il faut essentiellement remettre les bouts dans le bon ordre.

CAMILLE: Tu as raison, j'ai tout ce que je veux pour commencer à coder l'approximation et à l'occasion on reparlera de l'évaluation de performance. Tu as évoqué ça la dernière fois, ça m'intéresse.

CHARLIE: Pas de soucis, on en reparle. Je t'envoie de suite le programme Python avec les différents tests que l'on vient de voir.

Question 5. *Implémentez la 2-approximation, testez là sur des exemples simples puis sur des graphes générés aléatoirement.*

Votre deux approximations doit renvoyer une liste d'arêtes (des couples de sommets).

Inspirez-vous des fonctions de Charlie. Elles sont dans le fichier `test_networkx.py`*. Le fichier contient également une fonction permettant de vérifier qu'une solution est valide.*

2.6 Séries de graphes aléatoires

Maintenant que votre 2-approximation est codée, vous prenez rendez-vous avec Alix pour présenter vos premiers résultats. La date est fixée à la semaine prochaine. Vous préparez quelques slides pour illustrer la modélisation du problème et la 2-approximation mais vous n'êtes pas tout à fait satisfait de ce que vous rédigez. En particulier le rapport de 2 est finalement assez grand. Vous avez peur qu'Alix ne trouve pas ça terrible. C'est vrai qu'une solution valant 2 fois l'optimum ce n'est pas une si bonne nouvelle... Pour autant, Charlie avait l'air assez confiant, vous décidez donc d'aller le voir une fois de plus pour savoir comment il présenterait les choses.

CAMILLE: Mission accomplie, la 2-approximation est développée.

CHARLIE: Bravo ! Elle s'exécute rapidement ?

CAMILLE: Oui, finalement pas besoin de viser l'optimisation du code à tout pris. En tout cas pas pour une preuve de concept. Du coup, je vais abuser de tes conseils une nouvelle fois. J'ai peur que la 2-approximation ne soit pas satisfaisante pour Aerian Fiber. Je me dis que 2 fois l'optimum c'est quand même beaucoup.

CHARLIE: Oui, c'est beaucoup, mais attention, on parle du pire cas. Ce que garantit cette 2-approximation c'est que la solution approchée ne sera jamais à plus de 2 de l'optimum. En pratique, les solutions seront relativement proche de l'optimum. Il faut vraiment des instances tordues pour que l'on obtienne un rapport de 2. D'ailleurs c'est un bon exercice.

CAMILLE: De trouver une mauvaise instance ?

CHARLIE: Oui. Parce que finalement, la preuve de la 2-approximation, nous dit seulement que les solutions sont à une distance au plus 2. Rien ne nous dit qu'il n'existe pas une autre preuve permettant de garantir que les solutions sont à distance 1.5 ou même moins. Par contre, si on trouve une mauvaise instance. Une instance sur laquelle notre algorithme d'approximation est mauvais, à une distance $2 - \epsilon$ par exemple. Alors ça voudra dire que notre borne est serrée [7]. Ça voudra dire que le meilleur rapport d'approximation qu'on pourra prouver pour cet algorithme est 2.

CAMILLE: Je vais me mettre en quête d'une telle instance. Il n'y avait rien dans tes cours ?

7. en anglais, tight bound

CHARLIE: Non, le prof n'en avait pas parlé. On en avait parlé pour d'autres problèmes mais pas pour celui là.

CAMILLE: En tout cas, cette mauvaise instance, elle est importante pour moi parce que ça m'aidera à savoir si la preuve du rapport est bonne. Par contre ça ne va pas m'aider à convaincre Alix que mon algo donne de bons résultats en pratique.

CHARLIE: C'est vrai. Par contre, ce qui pourra aider c'est de générer un grand nombre d'instances aléatoires, d'exécuter ton algorithme et de regarder en moyenne son comportement. Par exemple, tu écris une fonction qui prend en paramètre une série de graphes et renvoie la valeur moyenne de la 2-approximation. Ensuite, tu génère des séries de graphes semblables. Par exemple des séries de n graphes, de taille s suivant un même modèle de génération aléatoire.

CAMILLE: Ok, et donc le but est de faire varier un à un les paramètres pour voir leur impact sur la qualité de l'algorithme ?

CHARLIE: C'est un peu l'idée. Par exemple, tu utilises le modèles de graphe aléatoire binomial et tu génère 100 graphes de taille 10, 20, 50, 100 et 200. À chaque fois tu calcul le poids moyen de la solution approchée. Ça permet de voir comment elle évolue avec la taille du graphe. Tu peux répéter l'opération sur des graphes qui ont toujours la même taille mais en faisant varier le nombre d'arêtes. Donc en augmentant la probabilité d'apparition d'une arêtes entre deux sommets ou en la diminuant.

CAMILLE: Alors faut pas trop la diminuer sinon générer des graphes aléatoires deviendra trop long vu qu'il faut qu'il soit connexe. Si la probabilité est trop faible, ça deviendra dur de tomber sur des graphes connexes.

CHARLIE: C'est vrai. Et ce qu'on vient de dire pour les graphes aléatoire binomiaux on peut le faire aussi pour d'autres types de graphes afin de voir ce que ça donne. Tu peux faire des jolies graphiques avec `matplotlib` pour illustrer tout ça. Ça fera un peu sérieux, ça devrait rassurer Alix.

CAMILLE: Oui, mais comment convaincre que les résultats sont bons ? Je veux dire, je vais faire un graphique avec le poids moyen des solutions mais il n'y pas de comparaison.

CHARLIE: Tu mets le doigt sur quelque chose de très important. Idéalement, lorsque l'on fait ce genre d'exercice il faut soit être capable de calculer les solutions optimales soit disposer d'une borne inférieure sur la solution optimale. Pour chaque série on peut alors calculer une référence, le poids moyen de la solution ou de la borne inférieure.

CAMILLE: Mais le problème est NP-complet, on ne peut pas calculer le poids de la solution optimale.

CHARLIE: Si, on peut. Il faut juste un peu de temps. Pour beaucoup de problème il existe des bases de données avec des instances de test. Pour ces instances on dispose des solutions optimales. Ça permet de tester de nouveaux algorithmes. Je vais regarder si j'en trouve pour ce problème. D'ici là, tu peux utiliser la seconde option. Trouve une borne inférieure.

CAMILLE: Pas simple. On sait que pour k terminaux, il faut au moins $k-1$ arêtes pour former un arbre. On pourrait faire une liste avec l'arêtes la moins cher de chacun des k terminaux. Si on supprime de cette liste l'arête la plus chère, on devrait avoir une borne inférieure. Mais je doute que ce soit pertinent.

CHARLIE: Ca semble effectivement très inférieure comme borne. C'est la configuration ou tous les terminaux sont connectés directement les uns au autres et en plus avec l'arête la moins chère à chaque fois. En fait ça ne prend pas en compte ce qui peut être couteux, à savoir les sommets intermédiaires. Quand on connecte deux terminaux ensemble, on doit parfois passer par des sommets qui ne sont pas terminaux.

CAMILLE: Pour ça on peut déjà regarder les deux terminaux qui sont les plus éloignées. Je veux dire, dans la solution optimale, il existe un chemin entre tout couple de terminaux. Et ce chemin est toujours de taille au moins aussi grande qu'un plus court chemin. Donc si je cherche la distance entre les deux terminaux les plus éloignés, j'ai une borne inférieure.

CHARLIE: Tout à fait. Et ça n'empêche pas de compléter cette borne avec les arrêtes évoquées tout au début. Ce chemin entre les deux terminaux les plus éloignés, on peut le voir comme la colonne verticale de l'arbre que l'on cherche. Tous les autres terminaux doivent s'y connecter. Et on sait que pour ça, chaque terminal devra payer au moins, le prix de son arête la plus petite.

CAMILLE: Bingo ! Je pars la dessus. Peut être qu'il y a mieux mais c'est déjà beaucoup d'informations. J'ai besoin de mettre en pratique.

Vous repartez dans votre bureau avec de nouvelles idées : trouver une mauvaise instance, générer des séries de graphes, calculer des solutions approchées et des bornes inférieures moyennes. De quoi vous occuper d'ici au prochain rendez-vous avec Alix.

Question 6. *Trouvez la "meilleure" des pires instances possibles. Essayez de construire une instance dont le rapport entre la solution fournit par la 2-approximation et la solution optimale est la plus mauvaise possible.*

Intégrez cette instance à votre rapport, à la suite de la preuve du rapport d'approximation.

Question 7. *Écrivez une fonction permettant de générer des séries de n graphes de taille s suivant un modèle aléatoire donné.*

Question 8. *Écrivez une fonction qui prend en paramètre une série de graphe et renvoie la solution moyenne.*

Question 9. *Écrivez une fonction qui prend en paramètre plusieurs séries de graphes variant selon un paramètre choisi. Votre fonction affiche sur une courbe la valeur de la solution moyenne en fonction du paramètre choisi.*

Question 10. *Écrivez une fonction qui calcule la borne inférieure de la solution optimum. Écrivez une fonction qui calcule la borne inférieure moyenne pour une série de graphes. Intégrez la valeur de la borne inférieur à la fonction qui fabrique la courbe.*

À suivre...