Université Paris 6 06 Mars 2018
MASTER ANDROIDE M1 – RP

Projet de résolution de problèmes :

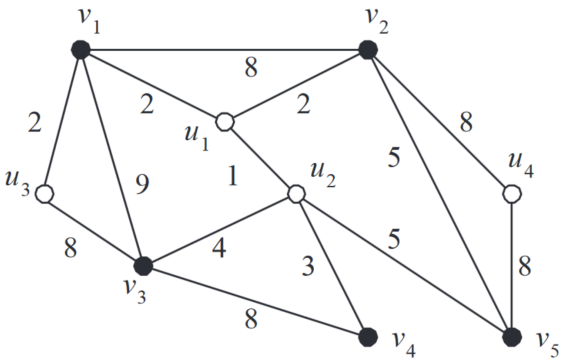
Metaheuristiques pour la résolution du problème de l'arbre de Steiner de poids minimum

Introduction et objectifs

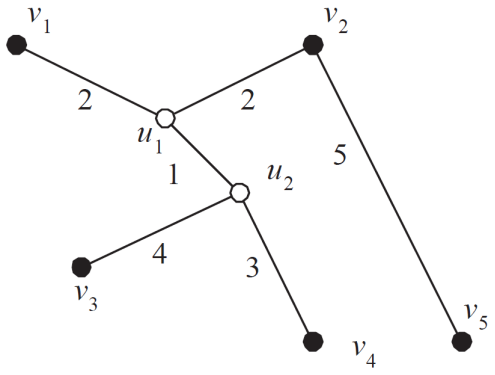
Le problème de l'arbre de Steiner de poids minimum est un problème d'optimisation combinatoire connu pour ses multiples applications dans le domaine de la conception de grands réseaux (télécommunication, distribution électrique, pipelines, ...), de circuits VLSI. Il existe plusieurs variantes de ce problème. Nous nous intéressons ici à la version graphique qui peut se résumer comme suit : étant donné un graphe G connexe non-orienté dont les arêtes ont des poids et un sous-ensemble T de sommets de G, trouver un ensemble d'arêtes de poids minimal tel que le sous-graphe induit soit connexe et contienne tous les sommets de T. Il s'agit donc de trouver un arbre de poids minimum reliant les sommets de T (appelés *sommets terminaux*). Contrairement au problème de l'arbre couvrant de poids minimum où tous les sommets de l'arbre doivent être dans T, dans le problème de l'arbre de Steiner il est autorisé d'utiliser des sommets qui ne sont pas dans T (appelés *sommets non-terminaux*). Plus précisément le problème se pose comme suit :

PROBLÈME DE L'ARBRE DE STEINER DE POIDS MINIMUM
INSTANCE : $G = (V, E)$ un graphe connexe valué par une fonction de poids $w : E \to \mathbb{N}$ et $T \subset V$ un ensemble non-vide de sommets (sommets terminaux).
QUESTION : Déterminer un arbre $G' = (V', E')$ tel que $T \subseteq V' \subseteq V$ et $w(E') = \sum_{e \in E} w(e)$ est minimum.

Exemple 1 *Considérons par exemple le graphe $G = (V, E)$ représenté ci-dessous, avec $V = \{v_1, v_2, v_3, v_4, v_5, u_1, u_2, u_3, u_4\}$ et $T = \{v_1, v_2, v_3, v_4, v_5\}$:*



Une solution optimale pour cette instance consiste à utiliser l'ensemble de sommets non-terminaux $S = \{u_1, u_2\}$ (appelés points ou sommets de Steiner*). Cette solution coûte 17 et conduit au graphe G' suivant :*



Le problème de l'arbre de Steiner de poids minimum est NP-difficile. Uniquement dans quelques particuliers, le problème de Steiner de poids minimum peut être résolu en temps polynomial : par exemple, si le nombre de sommets terminaux est égal à 2, le problème se réduit à un simple problème de plus court chemin et peut être résolu avec l'algorithme de Dijkstra. De même, si $V = T$, le problème est équivalent au problème de l'arbre couvrant de poids minimum et on peut utiliser l'algorithme de Prim ou Kruskal. Si l'ensemble S des sommets de Steiner était connu, l'arbre de Steiner de poids minimum serait simplement obtenu en calculant l'arbre couvrant de poids minimal du sous-graphe de G induit par $S \cup T$. Mais cet ensemble de Steiner n'est pas connu a priori.

Divers algorithmes approchés existent pour trouver des bonnes solutions, avec ou sans garantie sur la qualité des solutions trouvées. L'objet du projet est ici de tester quelques métaheuristiques pour ce problème, en particulier de comparer les résultats obtenus par un algorithme génétique et une méthode de recherche locale. Des indications pour le codage de ces deux méthodes vous sont donnés dans la suite du document, mais vous êtes libres d'utiliser vos propres opérateurs.

1 Un simple algorithme génétique

La première méthode que l'on vous propose d'explorer pour résoudre le problème de l'arbre de Steiner de poids minimum est basée sur les algorithmes génétiques.

1.1 Codage d'un individu

Un codage naturel pour représenter un individu peut se faire sous forme de chaîne de bits de taille égale à $|V|$ - $|T|$, où chaque bit à une position i correspond à un sommet non-terminal u_i, avec $u_i = 1$ si $u_i \in V'$ et 0 sinon. Par exemple, la chaîne de bits correspondant à la solution optimale de l'exemple 1 est égale à [1100]. Notez qu'il n'est pas nécessaire d'intégrer les sommets terminaux dans le codage car ils doivent nécessairement faire partir d'une solution admissible.

1.2 Evaluation d'un individu (fitness)

Pour évaluer la *fitness* f_i d'un individu i, vous pouvez vous considérer son coût obtenu de la manière suivante. Dans un premier temps, construisez le sous-graphe induit par les sommets terminaux et non-terminaux sélectionnés. Appliquez ensuite l'algorithme de Kruskal sur ce sous-graphe (tri des arêtes du graphe par poids croissant et sélection d'une arête si elle ne crée pas de cycle). Si le graphe est connexe, l'algorithme renvoie un arbre couvrant de poids minimum contenant $(|V'| - 1)$ arêtes. Dans ce cas, vous pouvez prendre comme fonction coût l'évaluation obtenue pour cet arbre. Par contre, si le graphe n'est pas connexe, la solution n'est pas admissible et l'algorithme renvoie une forêt composée d'arbres de poids minimum couvrant chaque composante

connexe. Dans ce cas, le coût de la solution sera défini comme le poids de la forêt plus une pénalité dépendant du nombre d'arêtes manquantes dans la forêt pour constituer un arbre couvrant les sommets de V', c'est-à-dire $w(F) + M \cdot (|V'| - 1 - a)$, où $w(F)$ représente le poids de la forêt F, a le nombre d'arêtes présentes dans F et M un grand nombre pénalisant une solution non admissible.

1.3 Opérateur de sélection et croisement

La sélection des parents pour l'opérateur de croisement doit se faire de manière proportionnelle à la qualité des individus : vous pouvez utiliser un process de sélection où la probabilité de sélection d'un individu i décroît avec le coût de cet individu tel que défini précédemment. Les parents sélectionnés subissent ensuite un croisement. Vous pouvez utiliser comme opérateur de croisement un opérateur à un point (comme vu en cours), qui produit deux nouveaux individus (les enfants).

1.4 Mutation

Un opérateur de mutation changeant aléatoirement un bit d'un individu enfant peut être appliqué (avec une faible probabilité, généralement entre 1 et 4%).

1.5 Remplacement et nouvelle génération

Deux stratégies de remplacement de l'ancienne population sont souvent considérées :

— Remplacement générationnel : la nouvelle population d'individus est entièrement composée des nouveaux individus issus du croisement.

— Remplacement élitiste : la nouvelle population d'individus est constituée des meilleurs individus parmi les parents et enfants.

Vous pourrez tester ces deux processus et sélectionnerez le schéma de remplacement donnant les meilleurs résultats.

1.6 Population initiale

Vous pouvez générer une population initiale de solutions où les solutions sont générées aléatoirement en fonction d'un paramètre p, donnant la probabilité qu'un bit soit égal à 1 chez un individu. Vous pouvez utiliser différentes valeurs de p pour générer des individus possédant différentes caractéristiques, et ainsi augmenter la diversité de la population (prendre par exemple des valeurs de p entre 0.2 et 0.5).

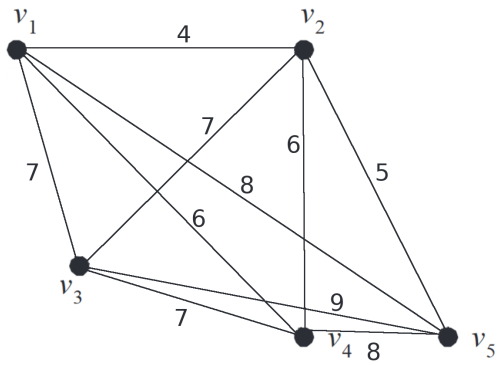
2 Amélioration de la population avec heuristiques de construction

L'utilisation de bits générés aléatoirement pour la génération de la population initiale risque de donner une population initiale de faible qualité et une lente évolution de la qualité des populations successives de l'algorithme génétique. On vous propose donc d'améliorer la qualité de la population initiale en utilisant des heuristiques de construction pour le problème de l'arbre de Steiner de poids minimum.

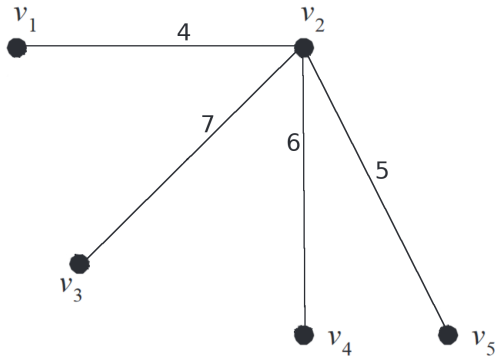
2.1 Heuristique du plus court chemin

L'heuristique du plus court chemin est une heuristique très efficace pour l'obtention d'une solution de bonne qualité (solution qui est une $2 - 2/|T|$ approximation de l'arbre de Steiner de poids minimum). Son fonctionnement est le suivant :

1. Construire un graphe complet $G_1 = (V_1, E_1)$ avec $V_1 = T$ et, pour chaque $(v_i, v_j) \in E_1$, la fonction de poids $w(v_i, v_j)$ est égale à la distance d'un plus court chemin entre v_i et v_j dans G. Ce graphe est appelé graphe de distance et est représenté ci-dessous pour le problème de l'exemple 1.



2. Trouver un arbre de poids minimum G_2 couvrant les sommets de G_1. Pour le graphe G_1, on obtient par exemple l'arbre G_2 suivant :



3. Construire le sous-graphe G_3 de G en remplaçant chaque arête de G_2 par les arêtes situées sur le plus court chemin correspondant dans G (pour l'exemple 1, cette opération permet d'obtenir directement l'arbre de Steiner optimal).
4. Construire l'arbre de poids minimum G_4 couvrant les sommets de G_3.
5. L'arbre de Steiner est alors le graphe G_5 que l'on obtient à partir de G_4 en éliminant, si nécessaire, des arêtes et des sommets de G_4 de telle façon qu'aucune feuille de G_5 ne soit un sommet non-terminal (dans la solution optimale, seuls les sommets terminaux ont un degré 1).

2.2 Heuristique de l'arbre couvrant minimum

L'heuristique de l'arbre couvrant minimum est une heuristique moins efficace que celle du plus court chemin, mais qui va permettre de diversifier la population de l'algorithme génétique. Son fonctionnement est le suivant :

1. Construire un arbre couvrant de poids minimum du graphe original $G = (V, E)$. Si toutes les feuilles de l'arbre sont des sommets terminaux, l'algorithme retourne l'arbre obtenu. Sinon, tous les sommets non-terminaux de degré 1 sont éliminés de l'arbre et un nouvel arbre de poids minimum du graphe induit par G en prenant les sommets restants après l'élimination est généré.
2. Répéter l'étape 1 jusqu'à ce que plus aucune élimination ne soit possible.

2.3 Randomisation des heuristiques de construction

Les deux heuristiques de construction étant déterministes, elles ne peuvent générer qu'au mieux deux solutions différentes. Dans le but de pouvoir utiliser ces heuristiques pour la génération d'une population d'individus, il est nécessaire de randomiser ces heuristiques. Un moyen simple est de perturber les données de départ : chaque poids associé aux arêtes est légèrement perturbé de manière aléatoire (en changeant par exemple leur poids de 5 à 20%). Ainsi, pour chaque création d'individu, une nouvelle instance est générée et les heuristiques de construction pourront donner des individus différents.

Vous comparerez les résultats obtenus avec la génération aléatoire de la population et la génération basée sur la randomisation des heuristiques de construction.

3 Recherche locale

Une seconde méthode que l'on vous propose d'implémenter est une méthode de recherche locale. Partant d'une solution initiale (qui peut être générée par une des heuristiques de construction), vous tenterez d'améliorer la solution grâce à une fonction de voisinage. La fonction de voisinage peut être définie de la façon suivante.

Soit une solution caractérisée par l'ensemble S de ces sommets non-terminaux et $T(S)$ l'arbre de Steiner associé. Les voisins d'une solution $T(S)$ sont définis par tous les ensembles de sommets non-terminaux qui peuvent être obtenus par ajout à S d'un nouveau sommet non-terminal (mouvement d'insertion) ou par élimination de S d'un de ses sommets non-terminaux (mouvement d'élimination). Il y a exactement un seul mouvement possible avec chaque sommet non-terminal. Pour un mouvement d'insertion, vous prêterez attention aux choses suivantes :

- Si le sommet s candidat à l'insertion n'est connecté à aucun sommet de $S \cup T$ alors le sommet peut être ignoré étant donné que le graphe induit dans G par $S \cup T \cup \{s\}$ ne sera pas connexe.
- Si s est connecté aux sommets de $S \cup T$ par uniquement une arête, il peut être ignoré (étant donné qu'il sera de degré 1).

A chaque mouvement d'insertion ou d'élimination, l'arbre couvrant de poids minimum devra être reconstruit.

Vous appliquerez la fonction de voisinage jusqu'à ce que plus aucun mouvement permettant d'améliorer la solution ne soit possible (en sélectionnant le premier mouvement améliorant). Si la recherche locale converge rapidement, vous pourrez éventuellement l'appliquer de nouveau, en partant d'une autre solution initiale, jusqu'à ce que le temps d'exécution accordé à la recherche locale soit atteint.

4 Instances et évaluation

Vous testerez et comparerez vos algorithmes (algorithme génétique simple, amélioré et recherche locale) sur diverses instances du problème, publiées sur le site suivant :
`http://steinlib.zib.de/testset.php`.

Vous utiliserez en particulier les groupes d'instances B, C, D et E. Les instances du groupe B sont des instances de petite taille, comprenant entre 50 et 100 sommets, utiles pour tester vos algorithmes. Les instances C, D et E sont plus grandes et comprennent respectivement 500, 1000 et 2500 sommets. Pour chaque instance, l'évaluation des solutions optimales sont données.

Vous donnerez pour chaque algorithme implémenté et chaque instance testée :

- L'évaluation de la solution obtenue
- Son écart (en pourcentage) par rapport à l'évaluation optimale

— Le temps d'exécution (idéalement vous devez faire en sorte que le temps d'exécution soit le même pour chacun des algorithmes testés, et inférieur à 5 minutes).

Vous présenterez également pour quelques instances les courbes montrant l'évolution de qualité de la meilleure solution obtenue à chaque itération de vos algorithmes en fonction du temps d'exécution.

Notez que le problème de l'arbre de Steiner de poids minimum fait pour l'instant l'objet d'un concours : `https://pacechallenge.wordpress.com/pace-2018/`.

Pour le concours de la meilleure heuristique, il y a 200 instances, numérotées de 1 à 200. Les instances impaires sont publiques tandis que les instances paires sont maintenues secrètes. Les instances publiques peuvent être téléchargées ici :

`http://www.lamsade.dauphine.fr/~sikora/pace18/heuristic.zip`.

Si vous participez au concours, vous ne serez pas évalués sur les instances publiques, mais sur les instances secrètes (de grande taille), via le site `http://www.optil.io/optilion/problem/3025`. Si vous soumettez votre code sur cette plate-forme, votre algorithme sera évalué sur ces instances secrètes, avec un temps d'exécution de maximum 30 minutes pour chaque instance. Les gagnants seront ceux qui obtiennent les meilleures approximations en moyenne.

Pour ce projet, il n'est pas demandé de participer au concours mais vous pouvez le faire si vous le souhaitez. En cas de bons résultats, votre participation pourra donner lieu à un bonus (sur le site du concours, veillez à intégrer vos initiales et "RP" dans votre nom d'utilisateur que l'on puisse attester de votre participation). Attention, vous avez jusqu'au **1er mai 2018** pour participer au concours.

Organisation et dates

— le langage d'implémentation recommandé est Python, toutefois, si ce langage ne vous convient pas d'autres options sont éventuellement admissibles (Java ou C/C++). Vous êtes libres d'utiliser des librairies de gestion de graphes.

— les projets doivent s'effectuer en binôme (étudiants du même groupe pour faciliter les soutenances). Informez votre chargé de TD des binômes constitués dès que possible (Groupe du Jeudi : nadjet.bourdache@lip6.fr ; groupe du vendredi : thibaut.lust@lip6.fr). En cas de difficulté à constituer un binôme, une dérogation pourra être accordée à titre exceptionnel après demande motivée (par mail) auprès du chargé de TD avec cc au chargé de cours (patrice.perny@lip6.fr).

— le projet doit être rendu au plus tard le **14 mai 2018**. Votre livraison sera constituée d'une archive zip qui comportera les sources du programme et un rapport au format pdf (de préférence rédigé en LaTeX). Le plan du rapport suivra le plan du sujet et résumera les choix méthodologiques, les principales réalisations et les tests numériques. L'archive zip devra être adressée à votre chargé de TD avec copie à Patrice Perny.

— la soutenance du projet est prévue en salle TME le 18 mai 2018. Prévoyez de pouvoir faire une démonstration sur machine de vos programmes.