TP3 – Métaheuristiques

Ce TP s'inscrit dans le prolongement du TP1, et vise à implémenter d'autres méthodes de résolution pour le problème de l'ensemble indépendant maximum. On a vu dans le TP1 que les méthodes exactes (qui renvoient toujours un ensemble indépendant maximum) deviennent rapidement trop lentes quand les instances grandissent. On s'intéresse ici à des algorithmes efficaces, mais qui peuvent se tromper : ils vont renvoyer une solution aussi bonne que possible, mais peuvent ne jamais rencontrer une solution optimale.

1 Algorithme glouton revisité

Question 1. Télécharger l'archive du TP3. Cette archive contient un corrigé des fonctions du TP1 utilisées aujourd'hui.

Question 2. Observer les graphes `graphEL_toygraph3` et `graphEL_toygraph4`. Que renvoie l'algorithme glouton sur ces graphes ? Comment expliquer ce comportement ?

Pour éviter ce problème, on va plutôt considérer les sommets dans un ordre aléatoire avant d'appliquer l'algorithme glouton.

Question 3. Que font les fonctions `gen_perm` et `shuffle` ?

Question 4. À l'aide de ces fonctions, compléter la fonction `greedy`, qui prend un graphe G et un entier t, et renvoie le plus grand ensemble indépendant parmi t indépendants calculés avec des ordres aléatoires sur les sommets de G.

Question 5. Tester votre algorithme sur les graphes de `data/`. Pour `graphEL_rand_1000_16` et `graphEL_roadNet-TX`, quelles valeurs de t pouvez vous choisir pour répondre en moins d'une minute ?

2 Recherche locale

La recherche locale consiste à se déplacer de solution en solution en essayant d'améliorer la solution à chaque fois. Il existe différentes stratégies possibles pour faire une recherche locale, certaines déterministes (prendre la solution qui améliore la solution actuelle qui soit lexicographiquement minimum par exemple), d'autres randomisées.
Pour initialiser l'algorithme on va calculer un indépendant maximal par inclusion. On va ensuite implémenter une phase de l'algorithme de recherche locale. Soit X l'ensemble indépendant actuel.

— On tire un sommet x de X au hasard,
— On calcule les sommets A adjacents à aucun sommet de $X \setminus \{x\}$.

— On calcule un ensemble indépendant maximal Y par inclusion de A.
— Si Y a taille au moins 2, on remplace X par $(X \setminus \{x\}) \cup Y$.

Question 6. Montrer que l'ensemble obtenu est un indépendant au moins aussi grand que X.

Question 7. Compléter la fonction `IS_completion`, qui prend en entrée un graphe G, un ensemble indépendant IS, et une liste de sommets autorisés L, et qui construit gloutonnement un ensemble indépendant de G maximal par inclusion contenant IS et des sommets de L.

Question 8. Écrire une fonction `RL_step` qui prend en entrée un graphe, un ensemble indépendant maximal par inclusion et un de ses sommets, et exécute une étape de l'algorithme de recherche locale.

Question 9. Écrire une fonction `RL` qui prend en entrée un graphe et un entier t, et qui envoie l'ensemble indépendant obtenu après t itérations de `RL_step`. On pourra s'aider de la fonction `sample`.

Question 10. Tester votre algorithme sur les graphes de `data/`. Pour `graphEL_1000_16` et `graphEL_roadNet-TX`, quelles valeurs de t pouvez vous choisir pour répondre en moins d'une minute ? Comparer les performances à celles de l'algorithme glouton.

Question 11. Écrire une variante déterministe `RL_det` qui effectue des étapes de l'algorithme de recherche locale tant qu'il en existe une qui améliore la solution courante. Au bout de combien d'étapes l'algorithme s'arrête-t-il sur les graphes de `data/` ?

3 Recuit simulé

Dans l'algorithme de recherche locale, on ne conserve une solution que si elle est meilleure que la solution courante. On peut donc rester bloqués dans des maxima locaux, c'est-à-dire des ensembles indépendants qui sont plus grands que tous ceux qu'on pourrait obtenir après un pas de recherche locale. Le problème est que tous les maxima locaux ne sont pas forcément des ensembles indépendants maximum du graphe.
Pour sortir de ces maxima locaux, il faut donc pouvoir autoriser des changements qui détériorent la solution courante. Une heuristique possible est celle du recuit simulé : si on peut améliorer la solution, on l'améliore. Sinon, si on trouve un changement qui détériore la solution, on le choisit avec une probabilité qui dépend de la perte de qualité et du temps déjà écoulé. Plus la perte est grande, et plus le temps écoulé est long, plus la probabilité sera faible.
On considère l'opération de changement suivante. Soit X l'ensemble indépendant actuel et x un sommet de X.
— On supprime de X tous les sommets à distance au plus 2 de x.
— On calcule un ensemble indépendant maximal Y contenant X.

Question 12. Compléter la fonction `RS_step` qui prend en entrée un graphe, un ensemble indépendant et un de ses sommets, et renvoie l'ensemble indépendant obtenu après avoir effectué l'opération précédente.

Pour l'algorithme de recuit simulé, on effectue t fois les opérations suivantes :

— Pour chaque sommet de l'IS courant I, appliquer `RS_step`.
— Si l'ensemble obtenu J est plus grand que I, remplacer I par J. Sinon, remplacer I par J avec probabilité $e^{-\frac{|I|-|J|}{T}}$.
— Après avoir traité chaque sommet, remplacer T par $0.95T$.

À la fin, on renvoie le plus grand ensemble indépendant qu'on a rencontré.
On peut assimiler la variable T à une température : plus elle est grande, plus on peut facilement changer de solution (même si elle est moins bonne). Plus le temps avance, plus la température refroidit, et les détériorations deviennent moins probables. On se comporte alors comme une recherche locale (avec une opération différente).

Question 13. Comment choisir la valeur initiale de T en fonction du nombre d'itérations t choisi ?

Question 14. Implémenter l'algorithme de recuit simulé dans la fonction `RS`.

Question 15. Tester votre algorithme sur les graphes de `data/`. Quelles valeurs de t pouvez vous choisir pour répondre en moins d'une minute ? Comparer les performances à celles des algorithmes précédents.

Pour bien comparer les performances de nos algorithmes, il faudrait considérer une recherche locale où on utilise la même opération de modification que pour le recuit simulé.

Question 16. Reprendre les questions de la partie 4 avec cette nouvelle opération. Il faudra donc implémenter `RL_step2` et `RL2`, analogues de `RL_step` et `RL`.

Question 17. Comparer les solutions obtenues par `RS` et `RL2`. Comment expliquer ces résultats ?