INF6102 - Métaheuristiques appliquées au génie informatique

Hiver 2025

Devoir 1
Étude de marché

Auriane Peter–Hemon - 2310513
Félix Lamarche - 2077446

2 Mars 2025

Table des matières

1 Introduction

L'objectif du devoir est le suivant : étant donné un graphe G, on cherche à déterminer les communautés, i.e. les groupements de noeuds, qui maximisent la densité de connexions entre les individus d'un même regroupement. Pour cela, on cherche à maximiser la modularité. En reprenant les notations de l'énoncé, on a :

$$Q = \frac{1}{2M} \sum_{i=1}^{N} \sum_{j=1}^{N} \delta_{c_i c_j} (A_{ij} - \frac{k_i k_j}{2M})$$

Pour cela, nous avons décidé de nous baser sur la méthode LPAm+ [2].

2 Algorithme de recherche locale

L'algorithme de recherche locale est basé sur l'algorithme LPAM : pour chaque noeud, on regarde si l'intégrer à la communauté d'un de ses voisins augmente la modularité. Si au moins une intégration améliore la solution, on choisit celle qui induit la plus grande amélioration de la modularité. Sinon, le noeud conserve sa communauté initiale.
C'est donc un algorithme Hill Climbing : on choisit le meilleur voisin améliorant. Toutes les contraintes sont dures : on ne considère que des solutions réalisables.

La modélisation est la suivante :
- **Solution initiale :** initialement, chaque noeud est seul dans sa communauté et les communautés sont triées en ordre décroissante selon le degré du noeud de la communauté
- **Voisinage :** pour chaque noeud, on considère les communautés de ses voisins. Le voisinage de la solution est l'ensemble des combinaisons des communautés considérées pour chaque noeud.
- **Évaluation** : la valeur de chaque voisin correspond à la valeur de Q pour cette solution
- **Sélection** : on sélectionne à chaque étape le meilleur voisin améliorant
- **Critère d'arrêt** : lorsqu'il n'existe aucun voisin améliorant, la recherche locale s'arrête.

Le voisinage que l'on a défini n'est pas un voisinage connecté. En effet, la recherche locale ne permet pas la création de communautés. Si on part d'une solution aléatoire comprenant moins de communautés que la solution optimale, il ne sera pas possible de créer des communautés pour atteindre cette dernière. C'est la raison pour laquelle on choisit cette solution initiale : on est assuré que la solution optimale comporte moins de communautés que notre solution initiale, donc notre algorithme peut atteindre le maximum global. De plus, on fait une recherche perturbative, en essayant des solutions complètes successives, et incomplète, puisque l'on ne garantit pas d'atteindre le maximum global.

3 Gestion des minima locaux

L'algorithme LPAm tend à favoriser les communautés de même taille et à rester coincé dans des maxima locaux [1]. La limitation de la méthode est qu'on ne peut changer simultanément plusieurs noeuds de communauté. On se retrouve alors dans des situations où changer un seul noeud de communauté n'améliorerait pas Q, mais changer simultanément plusieurs noeuds d'une même communauté l'améliorerait.

Pour pallier à ce problème, on s'inspire de la méthode utilisée par l'algorithme LPAM+ : une fois qu'un minimum local a été atteint, on change notre voisinage en combinant des communautés pour s'en échapper au lieu de changer un noeud de groupe à la fois. Plus précisément, pour chaque communauté, on calcule la modularité obtenue en la fusionnant avec chacune de ses communautés voisines. Ensuite, on la combine avec son meilleur groupe voisin, celui améliorant le plus la modularité de notre solution. On effectue ce processus jusqu'à ce qu'aucune fusion ne permette d'améliorer la modularité et que l'on atteigne un nouveau minimum local. Ce voisinage permet d'explorer des solutions où changer un seul noeud à la fois de communauté réduirait la modularité, mais que changer plusieurs noeuds simultanément de groupe l'amé-

liorerait. Pour simplifier notre voisinage, nous combinons deux communautés complètement.

Cela permet bien d'échapper au maximum local. Pour le montrer, on reprend la preuve donnée dans [2]. Soient $(i_1, i_2, ..., i_k)$ un groupe de noeuds "bloquants" appartenant à la même communauté $c_{i_1...i_k}$. Ces noeuds sont "bloquants" étant donné que la modularité décroît si l'on change de communauté un des noeuds de ce groupe, mais en changer plusieurs simultanément pourrait augmenter la modularité du système. On veut s'autoriser à changer leurs labels simultanément tout en souhaitant qu'ils partagent toujours le même label. Si on réécrit Q en isolant cette communauté bloquante, on a :

$$Q = \frac{1}{2M}(\sum_{i,j \notin \{i_1,...,i_k\}} \delta(c_i c_j) B_{ij} - \sum_{i,j \in \{i_1,...,i_k\}} B_{ij}) + \frac{1}{M}(\sum_{i \in \{i_1,...,i_k\}} \sum_{j \notin \{i_1,...,i_k\}} B_{ij} \delta(c_{i_1...i_k} c_j))$$

avec

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2M}$$

Le nouveau label des noeuds bloquants est, pour maximiser Q :

$$c^{new}_{i_1...i_k} = argmax_c(\sum_{i \in \{i_1,...,i_k\}} \sum_{j \notin \{i_1,...,i_k\}} B_{ij} \delta(c_{i_1...i_k} c))$$

Si ce nouveau label est différent de l'ancien, la mise à jour est bien équivalent à fusionner cette communauté avec la communauté bloquante.

Ce second algorithme est également un algorithme de recherche locale de Hill Climbing, sélectionnant le meilleur voisin à chaque itération. La solution initiale de cette algorithme est la solution ayant atteint un minimum local de notre autre algorithme.

Le voisinage de la solution est la fusion des communautés partageant un voisinage.

- **Voisinage :** pour chaque communauté, on considère les communautés voisines (ie les communautés pour lesquelles il existe une arête ayant une extrémité dans la communauté testée et une extrémité dans cette communauté) Le voisinage de la solution est l'ensemble des solutions accessibles en fusionnant deux communautés voisines.
- **Évaluation** : la valeur de chaque voisin correspond à la valeur de Q pour cette solution
- **Sélection** : on sélectionne à chaque étape le meilleur voisin améliorant
- **Critère d'arrêt** : lorsqu'il n'existe aucun voisin améliorant, la recherche locale s'arrête

En notant $\Delta Q_{c_i c_j}$ le changement de modularité induit par la fusion des communautés c_i et c_j, notre algorithme de recherche locale donne l'algorithme suivant :

Algorithm 1: LPAm+

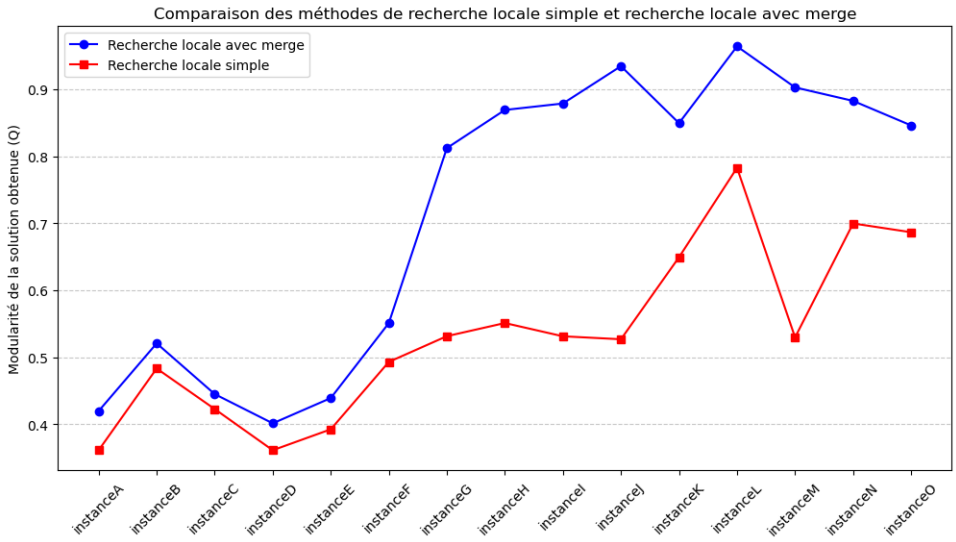
```
s = solutionInitiale()
while LPAm improves current solution do
    s = LPAm(s)
    while ∃ communities c_i, c_j with ΔQ_{c_i c_j} ≥ 0 do
        s = bestMerges(s)
    end while
end while
```

4 Résultats

On obtient les résultats suivants en temps d'exécution sur un CPU i7-8700k :

Pour se rendre compte de l'utilité de fusionner les communautés, on compare les résultats obtenus avec et sans fusion des communautés :

Instance	Q	Temps d'exécution (min)
instanceA	0.4198	0.0
instanceB	0.5208	0.0
instanceC	0.4451	0.00
instanceD	0.4015	0.00
instanceE	0.4392	0.00
instanceF	0.5510	0.04
instanceG	0.8120	0.01
instanceH	0.8689	0.01
instanceI	0.8785	0.05
instanceJ	0.9342	0.03
instanceK	0.8492	0.16
instanceL	0.9637	0.05
instanceM	0.9025	0.14
instanceN	0.8824	0.25
instanceO	0.8458	0.87



On remarque que la possibilité de fusionner les communautés augmente fortement la meilleure modularité obtenue. On parvient bien à s'extraire de maxima locaux.

5 Analyse de complexité

Voici une analyse succinte des fonctions principales du programme.

n : Nombre de noeuds dans le graphe.

Obtenir la solution initiale :
`__set_initial_groups()` : $O(n \log n)$.
La fonction trie l'ensemble de noeuds dans un ordre décroissant selon leur degré pour créer les groupes initiaux en créant un groupe par noeud.

Calculer le ΔQ de l'ajout d'un noeud d'un groupe à un autre :
`calculate_group_Q_with_node(neighbor_label, node` : $O(n)$.
Pour ne pas recalculer Q à chaque itération, on évalue uniquement la variation de Q causée par le déplacement d'un noeud. Pour l'ajout d'un noeud a dans une communauté c ayant c_n noeuds on n'a qu'à calculer

son apport à la modularité dans ce groupe, on a :

$$\Delta Q_c = -\frac{k_a^2}{2M} + \frac{1}{M}\sum_{i=1}^{c_n}(A_{ai} - \frac{k_a k_i}{2M})$$

Ensuite, on met à jour ΔQ en combinant le ΔQ_c de l'ancienne et de la nouvelle communauté. Cela est beaucoup plus rapide que de recalculer Q qui a une complexité asymptotique de $O(n^2)$.

Recherche locale avec LPAm :
`solve_LPAm()` : $O(n^2)$.
On itère pour chaque noeud, sur tous les noeuds présents dans ses groupes adjacents. Tant que l'on n'a pas atteint un minimum local, on itère : pour chaque noeud, on examine les groupes adjacents et on calcule le ΔQ avec la fonction `calculate_group_Q_with_node`. Ensuite, on déplace le noeud vers le groupe qui offre la meilleure amélioration de Q, s'il en existe un.

Calculer le ΔQ pour la combinaison de deux communautés :
`calculate_Q_of_merge(group_1_label, group_2_label)` : $O(n^2)$
La fusion de deux communautés donne la variation de Q suivante :

$$\Delta Q = \frac{1}{M}\sum_{i=1}^{n_{g1}}\sum_{j=1}^{n_{g2}}(A_{ij} - \frac{k_i k_j}{2M})$$

Tout comme pour le déplacement d'un noeud à une autre communauté, il est plus rapide de calculer le ΔQ que de recalculer entièrement Q. On parcourt chaque noeud de la communauté 1 et on l'évalue avec les noeuds de la communauté 2. On garde en mémoire les modularités Q_g de chaque groupe, ce qui évite de devoir recalculer les apports de modularité des noeuds dans les groupes eux-mêmes et que de calculer la nouvelle modularité provenant de la fusion des groupes et il peut y avoir jusqu'à $n/2 \times n/2$ calculs pour cette fusion.

Recherche locale pour fusionner les communautés :
`solve_greedy_merge()` : $O(n^2)$.
Tant que nous n'avons pas atteint le critère d'arrêt, on parcourt à chaque itération l'ensemble des groupes et de leurs groupes adjacents. Pour chaque groupe, on itère sur chacun des groupes adjacents afin de trouver la fusion apportant la meilleure amélioration de Q en calculant ΔQ avec notre fonction `calculate_Q_of_merge()`.
Ainsi, maximalement, l'algorithme est en temps $O(n^2$.

Recherche locale LPAm+ :
`solve_LMPAm_and_greedy` : $O(n^2)$.
La recherche locale continue jusqu'à ce qu'il n'y ait plus d'amélioration, donc le nombre d'itérations i est difficile à prédire. En pratique, dans les instances données, on restait en deçà de 10. Chaque itération a une complexité asymptotique égale au maximum entre les deux recherches locales utilisées (LPAm et la fusion de communautés), donc $O(n^2$.

Références

[1] Michael J. Barber et John W. Clark. "Detecting network communities by propagating labels under constraints". en. In : *Physical Review E* 80.2 (août 2009), p. 026129. issn : 1539-3755, 1550-2376. doi : 10.1103/PhysRevE.80.026129. url : https://link.aps.org/doi/10.1103/PhysRevE.80.026129 (visité le 20/02/2025).

[2] X. Liu et T. Murata. "Advanced modularity-specialized label propagation algorithm for detecting communities in networks". en. In : *Physica A : Statistical Mechanics and its Applications* 389.7 (avr. 2010), p. 1493-1500. issn : 03784371. doi : 10.1016/j.physa.2009.12.019. url : https://linkinghub.elsevier.com/retrieve/pii/S0378437109010152 (visité le 20/02/2025).