
Rapport du Projet de Bio-Informatique

Rémi Decouty et Damien Lu

1^{er} mars 2021

1 Présentation du sujet

On considère un ARN dont on veut en étudier la structure. L'idée est de considérer la structure d'un ARN comme étant un graphe, dont les noeuds sont les nucléotides et les arêtes du graphe sont les interactions entre les nucléotides.

Pour décrire ces interactions entre nucléotides, on considérera, conformément à l'article de Leontis, N.B. et Westhof, *Geometric nomenclature and classification of RNA base pairs*, la nomenclature de *Leontis-Westhof*.

On utilisera également la base de données RNANet contenant pour chaque chaîne d'ARN les différentes interactions entre les nucléotides de la chaîne.

L'objectif est d'implémenter un programme pouvant, à partir des fichiers constituant la base de données RNANet, générer un graphe représentant la structure de la chaîne d'ARN, puis à partir de motifs d'ARN préalablement choisis de rechercher les graphes contenant ces motifs d'ARN.

2 Choix techniques

Compte tenu du type de fichiers constituant la base de données RNANet (des fichiers CSV) et de leur manipulation plus aisée, nous avons choisi d'utiliser le langage Python. Celui-ci est d'ailleurs particulièrement bien adapté à ce projet, il dispose d'une librairie `python-igraph` permettant de représenter et manipuler des graphes.

3 Manuel d'utilisation

L'archive contient un fichier Python nommé `projet.py` qui contient le programme source, ainsi qu'un fichier `projet_multithread.py`, permettant de calculer le nombre de motifs total des RIN n°23 et 129 dans l'ensemble des fichiers CSV de RNANet. Pour optimiser l'usage du CPU, cette version n'affiche pas les graphes ni les messages dans la console, seulement le nombre de RIN calculé. Enfin, ce programme s'appuie sur le multithreading, pour plus de performance en terme de temps d'exécution.



Attention!

Le programme nécessite à minima Python 3.9 pour pouvoir correctement tourner.

Il nécessite également les modules Python suivants : `matplotlib`, `pycairo`, `pandas` et `igraph`. S'ils ne sont pas installés, veuillez exécuter la commande ci-dessous :

```
$ pip install matplotlib python-igraph pycairo pandas
```

Pour installer le projet, il convient de suivre les étapes suivantes :

1. Dézipper l'archive dans un répertoire que l'on appellera projet-bioinfo.
2. Télécharger l'archive `RNANET_datapoints_latest.tar` contenant les fichiers au format CSV depuis RNANet, à l'adresse suivante : <https://evryrna.ibisc.univ-evry.fr/evryrna/rnanet>, sélectionner "**DOWNLOAD RNANet Text files (CSV)**" et décompresser l'archive dans le dossier projet-bioinfo/data.
3. Exécuter le programme avec la commande :

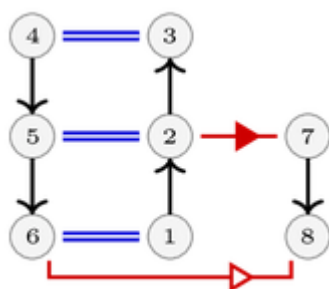
```
$ python projet.py
```

4 Explication du programme

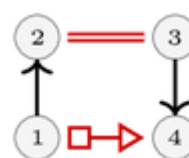
Le programme source du projet contient 4 fonctions principales :

- **color_edge(node, index)** : retourne la couleur de la liaison d'un nucléotide en fonction du type de celle-ci
- **draw_graph_from_csv(file)** : affiche un graphe modélisant le chaîne d'ARN, à partir d'un fichier au format CSV
- **find_subgraph(graph, motif, motif_name)** : retourne la liste des motifs trouvés dans le graphe
- **transorm_RIN_to_graph()** : modélise sous forme de graphes les RIN 23 et 129, puis les affichent

Dans ce rapport, nous allons présenter l'algorithme de recherche des sous-graphes d'un graphe modélisant une chaîne d'ARN, correspondant aux motifs RIN ci-dessous. De nombreux commentaires ont été ajoutés au code source pour les autres fonctions, afin de ne pas surcharger ce rapport.



(a) RIN n°23



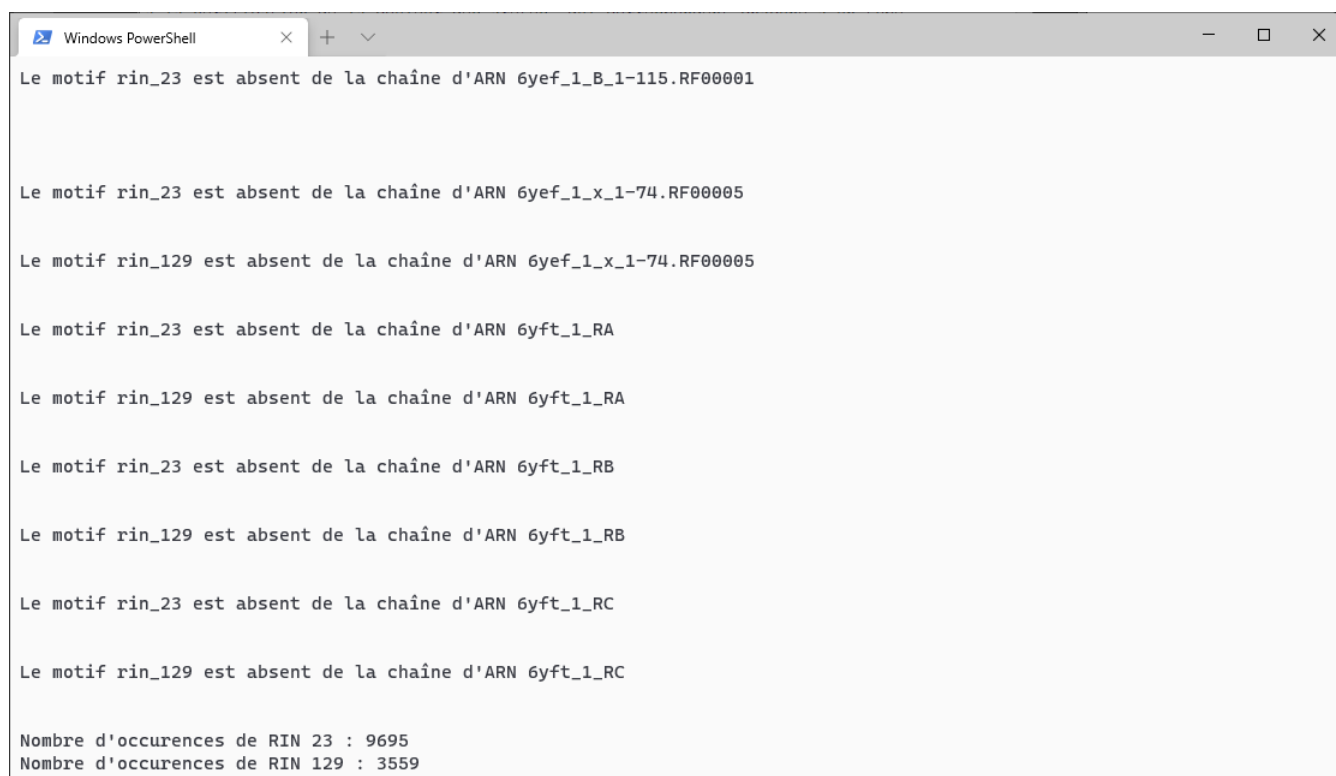
(b) RIN n°129

La fonction de recherche de motif, `find_subgraph`, fonctionne comme ceci :

1. La fonction `get_isomorphisms_vf2` issue de la librairie `igraph` permet de récupérer la liste de tous les sous-graphes isomorphes entre deux graphes. Ici, on va donc appliquer cette fonction aux graphes `g`, modélisant une chaîne d'ARN et au motif donné en argument.
2. Par ailleurs, pour restreindre la liste des sous-graphes, on va créer une fonction de comparaison des arêtes `compare_edges`. Celle-ci va simplement vérifier l'égalité de la couleur des arêtes issues de deux graphes, puis renvoyer le résultat sous forme de booléen.

3. On passe ensuite la fonction `compare_edges` en argument à la fonction `get_isomorphisms_vf2`. L'attribut `edge_compat_fn` permet de passer cette fonction de comparaison des arêtes. Pour plus d'informations, voir la documentation de la fonction ici : https://igraph.org/python/doc/igraph.GraphBase-class.html#get_isomorphisms_vf2
4. On a donc récupéré ici la liste des sous-graphes correspondant au motif donné, grâce à la vérification de la couleur des arêtes, qui correspondent chacune à un type d'interaction de Leontis-Westhof. Cependant, il reste encore un problème à résoudre : les doublons. En effet, le programme peut renvoyer le résultat : `[[0,1,2,3],[1,0,2,3]]`. On voit que les deux sous-graphes sont identiques, les indices ont simplement été déplacés. Pour résoudre ce problème, nous avons utilisé une la structure de données immuable `frozenset`, afin de stocker temporairement un élément de la liste. En fait, on va itérer sur chacun des éléments de la liste des sous-graphes, et s'assurer qu'aucun n'est identique à l'élément courant. On va ainsi supprimer de nombreux doublons, et donc réduire la taille de la liste des sous-graphes.
5. Pour finir, on a donc récupéré la liste des sous-graphes de `g`, correspondant au RIN passé en argument. Cependant, il reste un point que nous n'avons pas su résoudre. En effet, dans la liste des sous-graphes, il peut y avoir des éléments qui possèdent des noeuds en commun. Par exemple, on peut avoir : `[[0,1,2,3],[2,3,4,5]]`. Cela a pour conséquence, lorsque l'on calcule le nombre de RIN sur tous les fichiers, qu'on a un nombre très important de motifs trouvés, comparé à celui attendu sur Carnaval. Cela est normal, étant donné qu'on détecte parfois des motifs passant par un noeud identique. Il aurait dans l'idéal fallu être plus restrictif vis à vis de la détection de motif (aucun noeuds en commun pour deux motifs différents), mais nous n'avons malheureusement pas su mettre en oeuvre cette solution.

Pour illustrer le propos, voici le résultat de l'exécution de la version multithreadée du projet (commande `python projet_multithread.py`):



```
Windows PowerShell
Le motif rin_23 est absent de la chaîne d'ARN 6yef_1_B_1-115.RF00001

Le motif rin_23 est absent de la chaîne d'ARN 6yef_1_x_1-74.RF00005

Le motif rin_129 est absent de la chaîne d'ARN 6yef_1_x_1-74.RF00005

Le motif rin_23 est absent de la chaîne d'ARN 6yft_1_RA

Le motif rin_129 est absent de la chaîne d'ARN 6yft_1_RA

Le motif rin_23 est absent de la chaîne d'ARN 6yft_1_RB

Le motif rin_129 est absent de la chaîne d'ARN 6yft_1_RB

Le motif rin_23 est absent de la chaîne d'ARN 6yft_1_RC

Le motif rin_129 est absent de la chaîne d'ARN 6yft_1_RC

Nombre d'occurences de RIN 23 : 9695
Nombre d'occurences de RIN 129 : 3559
```

FIGURE 2 – Résultat de la commande `python projet_multithread.py`

À titre d'information, cette commande a mis environ 7 min à s'exécuter sur une machine dotée de 6 coeurs et 12 threads et environ 20 min sur une machine dotée de 4 coeurs et 8 threads. Il faut

donc prévoir selon la configuration de votre machine de quelques minutes à quelques dizaines de minutes pour l'exécution du programme.

5 Conclusion

Pour conclure ce rapport, nous allons faire un point sur ce que nous avons réussi à implémenter par rapport aux consignes ainsi que les points d'amélioration.

- Modélisation sous forme de graphe d'une chaîne d'ARN stockée au format CSV. Une amélioration possible consisterait à encadrer sur le dessin du graphe les motifs trouvés, avec des carrés rouge et vert, correspondant respectivement au RIN 23 et 129.
- Détection des motifs RIN, bien qu'encore améliorable. Il reste à supprimer les sous-graphes partageant au moins un noeud en commun, pour réduire la taille de la liste des sous-graphes.
- Parcours de l'ensemble des fichiers et calcul du nombre de RIN. Nous avons d'ailleurs constaté que ce calcul prenait beaucoup de temps sur un seul thread, c'est pour cela qu'une seconde version parallélisée du programme, nommée `projet_multithread.py`, est disponible. Vous constatez un grand nombre de RIN détectés (plusieurs milliers), ce qui est logique vu le problème des noeuds partagés entre sous-graphes, comme évoqué précédemment.