Projet IN608

CIESLA Julie, GODET Chloé, GROSJACQUES Marwane, HAMOUDI Nabil

Semestre 6 - 2023

Table des matières

1 Le sujet

1.1 Le choix du sujet

Nous avions trois possibilités de sujets.
- A general Edit distance between RNA structures [1]
- Fast algorithm for predicting the secondary structure of single stranded RNA [2]
- A Dynamic Programming Algorithm for RNA Structure Prediction Including Pseudoknots [3]

Nous avons choisi ce dernier sujet car nous n'avions jamais entendu parler des pseudonœuds en tant que structure secondaire de l'ARN auparavant. C'était donc l'occasion d'en apprendre plus sur le sujet, et de réellement mêler biologie et informatique.

Nous savions également que ce genre d'algorithme de prédiction est assez difficile, ce qui permet de nous mettre au défi et a permis d'accroître notre motivation.

1.2 Contexte

1.2.1 Objectif de l'algorithme

Le but du sujet de recherche d'Elena Rivas et Sean R. Eddy, auteurs de l'article, a été d'implémenter un algorithme de prédiction des structures secondaires d'ARN incluant la prédiction de la formation de pseudonœuds. Ils ont décidé d'utiliser la programmation dynamique.

Ils ont donc cherché à résoudre le problème en le décomposant en plusieurs sous-problèmes. Ils stockent les résultats intermédiaires lors de la résolution des sous problèmes et obtiennent une solution optimale.

Cet algorithme se base sur la minimisation de l'enthalpie libre afin de prédire la structure secondaire la plus stable possible.

1.2.2 La structure secondaire de l'ARN

Il existe plusieurs niveaux de structure dans l'ARN : primaire, secondaire, tertiaire et quaternaire. L'ARN, contrairement à l'ADN, se trouve souvent sous forme simple brin et peut donc plus facilement se replier sur lui-même grâce à des liaisons hydrogènes intramoléculaires. On y retrouve notamment les liaisons Watson-Crick (G-C, A-U) mais aussi un appariement appelé "wobble pairing" (G-U).

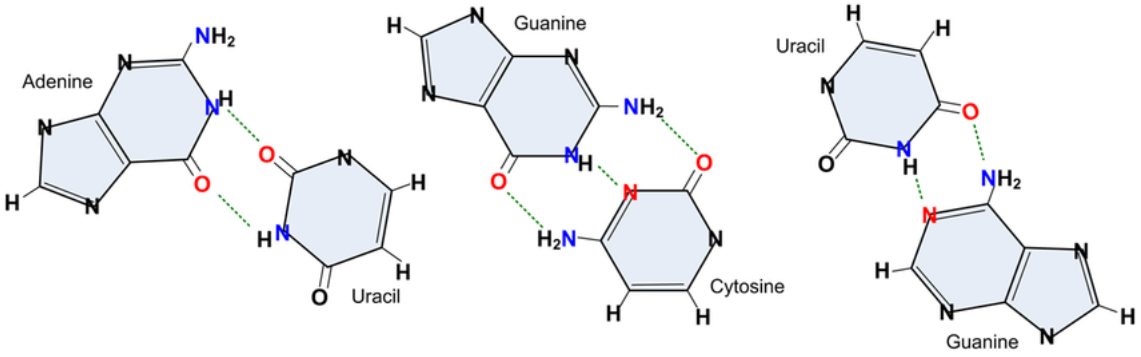


FIGURE 1 – liaisons A-U, C-G et U-G [4]

La structure secondaire de l'ARN correspond donc à la description des appariements internes au sein de la molécule, permettant la formation d'une structure stabilisée. Parmi les structures secondaires que l'on peut trouver, on peut citer l'épingle à cheveux, les boucles internes ou encore les boucles multiples.

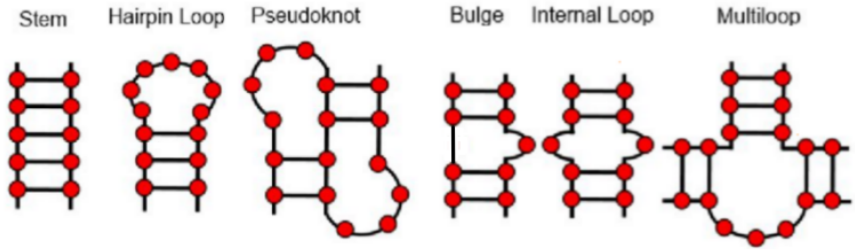


FIGURE 2 – structures secondaires de l'ARN [5]
stem = tige ; hairpin loop = boucle en épingle à cheveux ; pseudoknot = pseudonœud ; bulge = renflement ; internal loop = boucle interne ; multiloop = boucle multiple

Dans la suite de notre rapport, nous utiliserons volontairement ces termes en anglais.

Mais il existe également une structure secondaire particulière, dont on parle moins : **les pseudonœuds**. Cette dernière se forme à partir d'une structure tige-boucle, dont certains nucléotides de la boucle s'apparient avec une région en dehors de la tige. Les pseudonœuds sont importants pour la fonction de certains ARNs tels que des ARN ribosomaux. Cette structure est cependant peu fréquente.

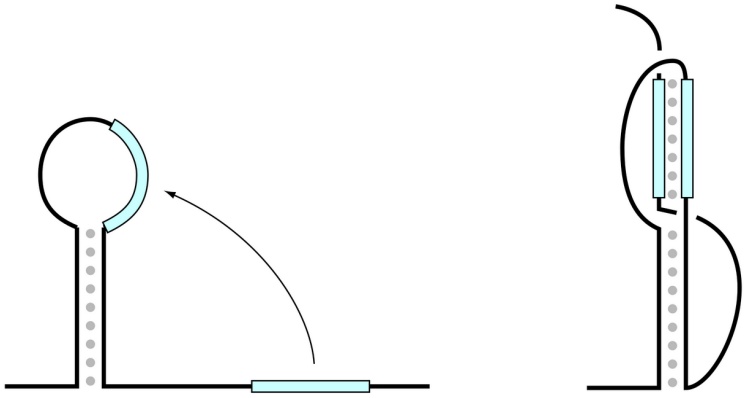


FIGURE 3 – Exemple d'une formation de pseudonœud [6]

1.2.3 La problématique

La prédiction des pseudonœuds étant considérée comme un problème NP-complexe (non-déterministe polynomial), c'est-à-dire que la prédiction requiert des calculs dont la complexité augmente exponentiellement avec la longueur de la séquence d'ARN, les auteurs de l'article ont dû faire des compromis.

En effet, l'algorithme ne prédit pas toutes les structures de pseudonœuds. Il prédit au moins trois topologies de pseudonœuds observées dans la nature (voir figures ci-dessous).

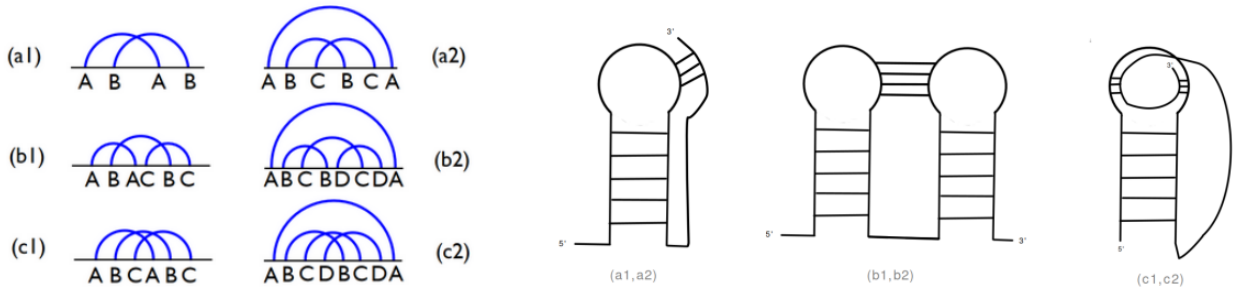


FIGURE 4 – Trois topologies de pseudonœuds : diagrammes de Feynman (gauche) et dessins (droite) [7] [8]

Ainsi ils ont développé un algorithme de complexité $O(n^6)$ en temps et $O(n^4)$ en mémoire contre $O(n^3)$ en temps pour des algorithmes sans prédiction de pseudonœuds tels que les algorithmes de Zuker-Sankoff et Nussinov dont se sont inspirés Elena Rivas et Sean R. Eddy.

2 Travail bibliographique

Dans un premier temps, il a fallu lire et comprendre l'article scientifique *A Dynamic Programming Algorithm for RNA Structure Prediction Including Pseudoknots.*

L'article n'étant pas simple à la première lecture, il nous a fallu, à chacun, quelques jours afin d'intégrer toutes les informations nécessaires à notre compréhension.

Nous avons également dû nous familiariser avec les diagrammes de Feynman que, jusque-là, nous n'avions jamais rencontrés.

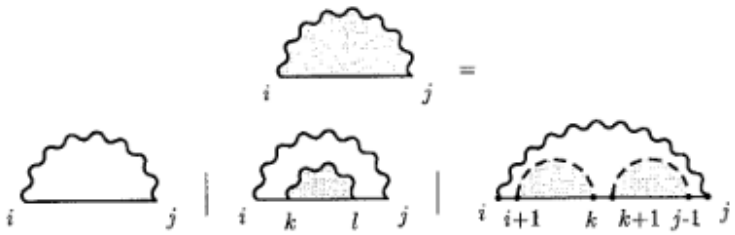


FIGURE 5 – Exemple d'un diagramme de Feynman (figure 6 de l'article) [9]
ligne continue : nucléotides appariés, ligne discontinue : indéterminé

De plus, comme mentionné dans Le choix du sujet, nous ne connaissions pas les pseudonœuds. Nous avons donc, chacun de notre côté, cherché des informations à ce sujet.

2.1 Les paramètres thermodynamiques

L'article [3] donne certaines informations concernant les matrices, comme ce qu'elles représentent, leur initialisation ou encore comment elles doivent être remplies. Cependant, elle ne donne pas d'énergie chiffrée.

La prédiction de la structure secondaire de l'ARN, dans notre cas, se fait à partir de paramètres thermodynamiques. L'algorithme génère une structure optimale (stable) pour une séquence d'ARN simple brin, en utilisant ces paramètres ainsi que des paramètres spécifiques décrivant la stabilité des pseudonœuds. Ces paramètres sont donc essentiels pour le fonctionnement de l'algorithme.

Malheureusement, nous retrouvons à plusieurs reprises dans l'article, des phrases comme : "For the relevant nested structures, hairpin loops, bulges, stems, internal loops and multiloops, we have used the same set of energies as used in MFOLD". soit en français : "Pour les structures imbriquées pertinentes, les boucles en épingle à cheveux, les renflements, les tiges, les boucles internes et les multiboucles, nous avons utilisé le même ensemble d'énergies que celui utilisé dans MFOLD."

Cela a posé un réel problème car aucun lien de l'article ne fonctionnait. Nous nous sommes donc demandé comment nous allions pouvoir récupérer ces données (ou d'autres similaires). Nous nous sommes rendus sur le site mfold.org mais n'avons pu les trouver. Une grande partie de notre temps de recherche du mois de Mars s'est donc concentrée sur la recherche d'autres articles pouvant contenir ce qui nous intéressait.

L'article *"Improved free-energy parameters for predictions of RNA duplex stability"* [10] et le site *Nearest Neighbors DB* [30] nous ont été très utiles. Ces derniers contiennent beaucoup de valeurs de paramètres qui nous étaient nécessaires et nous a permis d'avancer. C'est donc principalement avec ces valeurs que notre algorithme fonctionne.

2.2 Les récursions

De plus, les récursions pour les matrices n'étaient pas complètes. Nous pouvons lire dans l'article [3] "In order to give the reader a feeling for the kind of topologies the pseudoknot algorithm allows, we provide in the Appendix a simplified version of the recursions for the gap matrices in which coaxial stacking or dangles are excluded." soit "Afin de donner au lecteur une idée du type de topologies que l'algorithme incluant les pseudonœuds permet, nous fournissons en annexe une version simplifiée des récursions pour les matrices dans lesquelles le coaxial stacking ou les dangles sont exclus."

Ceci est un autre problème car, si leur initialisation renseignée dans cet article n'a pas posé de problème, les récursions manquantes nous auraient empêché de tester notre algorithme et de comparer nos résultats avec ceux des auteurs. Sans les dangles et le coaxial stacking, nos résultats auraient été forcément faussés et certaines de nos fonctions inutiles. Nous avons fini par trouver l'article *"complete set of recursions"* sur le site : Eddy Lab : Publications [11]. Il contient l'ensemble des récursions nécessaires à la réalisation de notre programme. Nous avons pu compléter nos matrices avec ces dernières.

2.3 Le reste de l'algorithme

2.3.1 Que faire une fois les matrices remplies ?

Notre article ne donne aucune information supplémentaire concernant l'algorithme.

Nos matrices contiennent donc des énergies et nous devons pouvoir retrouver non seulement le meilleur score (donc l'énergie la plus faible) mais déterminer également la façon dont on l'obtient.

2.3.2 Comment représenter les résultats ?

Enfin, nous ne savions pas quel est le résultat exact retourné par le programme des auteurs (meilleur score, structures, matrices ?). Nous avons donc choisi ce résultat en affichant ce qui nous paraissait important.

Nous savons que les résultats ne sont pas forcément simples à lire et à comprendre et que cela est souvent plus facile lorsque ceux-ci sont visuels. Afin de simplifier cette lecture, nous avons cherché une façon de dessiner les structures secondaires obtenues.

Il nous fallait d'abord déterminer les différentes structures de l'ARN, c'est pourquoi nous avons opté pour le Format DBN. Cela sera utile puisque nous l'utilisons lors de la création de La grammaire formelle.

Avec le temps imparti, nous savions que nous ne pourrions pas coder un programme permettant de dessiner les structures de façon propre et fonctionnelle. Nous avons donc choisi d'utiliser VARNA [12], une application de visualisation de la structure secondaire de l'ARN codée en JAVA.

3 Notre algorithme

3.1 Les matrices

Tout d'abord, il a fallu recenser l'ensemble des matrices nécessaires, comprendre leur utilité, leur contenu et réfléchir à leur implémentation.

Nous avons créé trois matrices 2D : vx, wx, wxi ainsi que quatre matrices 4D : vhx, whx, yhx et zhx.

Matrices (i ≤ k ≤ l ≤ j)	Relation i, j	Relation k, l
vx(i,j)	appariés	—
wx(i,j)	indéterminée	—
vhx(i,j : k,l)	appariés	appariés
zhx(i,j : k,l)	appariés	indéterminée
yhx(i,j : k,l)	indéterminée	appariés
whx(i,j : k,l)	indéterminée	indéterminée

FIGURE 6 – Tableau de la description des matrices vx, wx, vhx, whx, yhx et zhx [13]

Les matrices wx et wxi sont similaires, leur initialisation est identique. Cependant, wxi est utilisée lorsque l'on se trouve au sein d'une multiloop contrairement à wx. Elles ont donc les mêmes équations mais avec des paramètres thermodynamiques différents.

Nous avons choisi de créer des matrices triangulaires afin de limiter l'utilisation superflue de la mémoire.

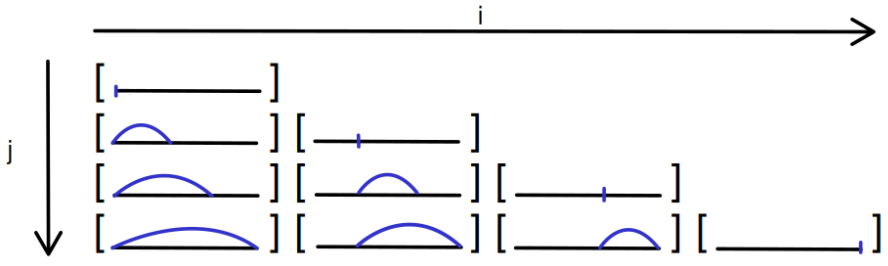


FIGURE 7 – Représentation d'une matrice triangulaire 2D [14]
les arcs de cercle bleus représentent la relation entre i et j

3.2 Les paramètres

Certains paramètres sont donnés dans la table 2 [15] et 3 [16] (spécifique des pseudonœuds) de l'article. Nous avons stocké ces valeurs dans :

- quatre dictionnaires : trois correspondent à l'énergie en fonction de la taille de la structure (bulge, hairpin loop et internal loop) et le dernier regroupe tous les paramètres qui sont utilisés dans Les récursions
- douze matrices : six pour les stacking (deux paires de bases donc appariées) et six pour les terminal-mismatch stacking (une paire de base et deux bases qui leur sont contiguës non appariées)

Nous avons codé treize fonctions permettant le calcul de l'énergie à partir de certains de ces paramètres.

3.2.1 Les Dangles

Des nucléotides non appariés contigus à une paire de base contribuent différemment (en termes de thermodynamique) comparé aux autres nucléotides non appariés, à la stabilité de l'ARN. C'est ce qu'on appelle les dangles. Ces dangles peuvent être à gauche ou à droite.

(a) Dangle à gauche (b) Dangle à droite

FIGURE 8 – Représentation d'un dangle [17]

On peut également trouver plusieurs dangles dont certains ne se trouvent pas forcément aux extrémités terminales mais peuvent être à l'intérieur d'une structure.

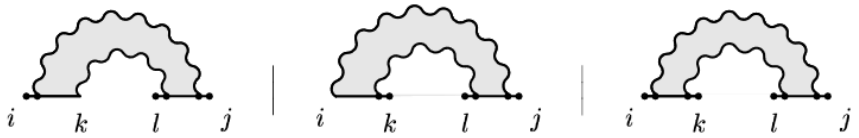


FIGURE 9 – Représentation de structures comportant plusieurs dangles [18]

L'énergie de chaque dangle a donc été renseignée dans ces deux fonctions (dangle gauche et dangle droit). Nous avons créé 4 autres fonctions de dangles car l'enthalpie libre change en fonction de l'environnement. Ainsi, il fallait deux fonctions pour les dangles au sein d'une multiloop, et deux de plus pour ceux au sein des pseudonœuds.

3.2.2 Coaxial stacking

Deux structures indépendantes sur le même axe peuvent se replier l'une sur l'autre. C'est ce que l'on appelle le **coaxial stacking** (ou "repliement coaxial" en français).

Le coaxial stacking se produit lorsque ces deux structures secondaires d'ARN proches forment une hélice en se repliant, qui est stabilisée par un empilement de bases se trouvant aux extrémités des deux

structures secondaires.

Il existe deux types de coaxial stacking : “flush” (les deux bases nucléotidiques terminales sont appariées de chaque côté) et “mismatch-mediated” (il existe un appariement pour deux bases d’un côté et un mismatch de l’autre).

```
5'CA–CA 3'
   || ||
3'GU GU 5'
   G A
   5'3'
```

FIGURE 10 – Flush [19]

```
5'CAG–CA 3'
   ||   ||
3'GUG GU 5'
   G A
   5'3'
```

FIGURE 11 – Mismatch-mediated [20]

Seul le coaxial stacking "flush" est pris en compte dans l’algorithme. Dans ce cas, les paires terminales de deux structures contiguës donnent alors la même stabilité que si elles avaient formé un stacking (un empilement) entre elles.

Ainsi la formation de ces hélices permet de réduire l’énergie de la structure secondaire de l’ARN.

Comme pour les dangles, nous avons eu besoin de deux fonctions. Ces fonctions font appel à la fonction calculant l’énergie d’un stacking avec, chacune, un coefficient différent.

Il est important de noter que le coaxial stacking intervient lors de la formation de la structure tertiaire de l’ARN, par conséquent l’algorithme ne se limite pas seulement à la structure secondaire.

3.2.3 Les surfaces irréductibles

L’article introduit la notion de **surface irréductible**. Une surface irréductible est une surface pour laquelle, si une liaison hydrogène est brisée, il n’y a pas d’autre surface à l’intérieur. Elle ne peut donc pas “être réduite” à une autre surface.

Les appariements des pseudonœuds n’étant pas imbriqués, ils ne sont pas concernés par cette définition.

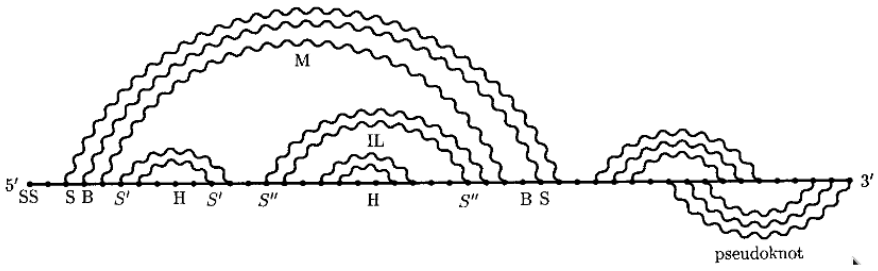


FIGURE 12 – Représentation des structures secondaires dans l’ARN [21]

H : hairpin loop, S : stem, IL : internal loop, B : bulge, M : multiloop, SS : single-stranded (non impliqués dans une structure)

Les hairpins (boucle en épingles à cheveux) sont des surfaces irréductibles d’ordre 1. Elles sont en effet bornées par une paire de bases.

Les bulges (renflement), internal loops (boucles internes) et stem (tiges) sont, quant à eux, des surfaces irréductibles d'ordre 2. Ces structures sont, en effet, bornées par deux paires de bases. On peut citer leur différence :

- **Stem** : ils sont bornés par deux paires de bases contiguës aux deux extrémités.
- **Bulges** : ils sont bornés par deux paires de bases contiguës à une seule extrémité
- **Internal loop** : ils sont bornés par deux paires de bases non contiguës.

Les multiloop (boucles multiples) sont des surfaces irréductibles d'ordre supérieur à 2 (elles sont bornées par au minimum 3 paires de bases). C'est pour cette raison que, si les récursions sont assez précises pour les surfaces irréductibles d'ordre 1 et 2, elles sont davantage approximatives pour les multiloop.

En effet, une surface irréductible d'ordre $\mathcal{O}(\gamma)$ ajoute une complexité en temps de $\mathcal{O}(N^{2(\gamma-1)})$ avec N la taille de la séquence d'ARN. Cela deviendrait donc impossible pour nous d'obtenir une solution.

Nous avons donc créé deux fonctions, une pour chaque ordre de surface irréductible. Ces dernières permettent d'obtenir l'énergie de la structure en fonction de la taille de celle-ci ainsi que de l'énergie des paires de bases terminales (bornant la structure). Une dernière fonction est présente pour calculer cette énergie quand la surface irréductible d'ordre 2 est incluse dans un pseudonœud.

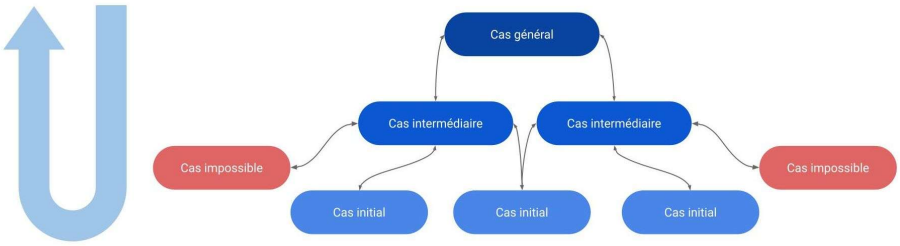
Ce sont ces paramètres et ces fonctions dont on se sert dans les récursions pour remplir les matrices.

3.3 Programmation dynamique

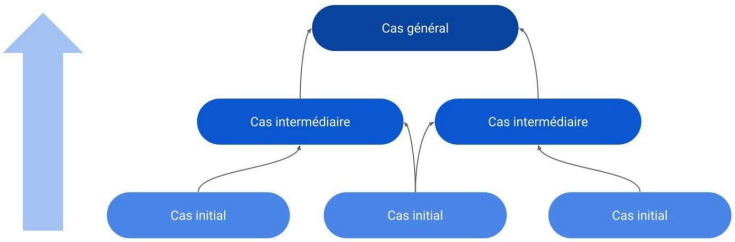
Les matrices décrites précédemment sont utilisées afin de garder en mémoire les résultats obtenus évitant ainsi de les recalculer.

Une première possibilité serait d'utiliser une approche récursive. Cette méthode intuitive est simple à implémenter en échange de performances réduites. Nous partirions donc du cas général pour arriver peu à peu vers les cas initiaux puis nous remonterions jusqu'au cas général en construisant et en gardant en mémoire les résultats au fur et à mesure.

Une autre possibilité plus performante consisterait à partir des cas initiaux directement et de remonter au cas général à l'aide d'une boucle. Cela demande plus de précision et donc plus de réflexion quant à l'implémentation. En effet, il faut s'assurer que les équations calculées à l'itération actuelle ne demandent pas des valeurs qui n'ont pas encore été calculées. Cette méthode ouvrirait la possibilité de parallélisme en plus d'éviter l'utilisation de récursivité ce qui donnerait de biens meilleures performances.



(a) méthode 1



(b) méthode 2

FIGURE 13 – Schéma des méthodes décrites ci-dessus [22]

Notre programme étant codé en langage python, les performances ne sont pas nos priorités. Nous préférons nous concentrer sur la clarté et l'intuitivité du code. C'est pour ces raisons que nous avons choisi la première méthode, celle de la récursion.

À chaque matrice est associée une fonction, prenant en argument des indices (deux pour les matrices 2D et quatre pour les matrices 4D), qui fonctionne de la manière suivante :

- Nous vérifions si les indices sont hors de portée, auquel cas nous retournons une valeur très grande
- Nous regardons ensuite si la valeur n'a pas déjà été calculée, sinon nous la retournons
- Nous calculons les équations à la suite desquelles nous prenons la valeur la plus petite que nous retournons après l'avoir gardée en mémoire dans la matrice appropriée.

Les équations feront donc elles même appel aux fonctions récursives des matrices.

3.4 Traceback

Comme expliqué dans Le reste de l'algorithme, le meilleur score et la façon de l'obtenir était une partie relativement floue.

Nous avons pensé à regarder comment cela était fait dans l'algorithme de Zuker, puisque l'article de recherche en est inspiré. Il s'avère que la technique utilisée, qui est la plus courante pour ce genre de choses, est le **traceback**.

Ayant déjà utilisé cette technique en cours de biologie sur de petites matrices, nous visualisions relativement bien comment faire.

3.4.1 La matrice WX

La matrice WX représente le cas le plus général. En effet, pour deux indices donnés, que les nucléotides à ces indices soient appariés ou non, cette matrice donne la plus petite énergie possible entre ces deux indices compris.

Nous en avons donc déduit que l'énergie de la structure optimale pour une séquence de taille N se trouvait dans la matrice WX aux indices $i = 0$ et $j = N - 1$. (voir figure 7)

3.4.2 Le traceback

Afin de déterminer quelles bases sont appariées avec quelles autres bases, il faut retracer le chemin qui nous a permis d'obtenir l'énergie minimale de la structure optimale.
Pour cela, deux choix récursifs étaient possibles :

- Recalculer toutes les équations d'une matrice pour des indices donnés afin de savoir quelle équation a permis de calculer l'énergie gardée en mémoire (donc l'énergie minimale) à ces indices. Il faut ensuite réitérer cela pour toutes les matrices à tous leurs indices utilisés dans l'équation.
- Garder en mémoire les matrices ainsi que leurs indices utilisés en même temps que nous calculons la valeur de l'énergie.

La première méthode est plus efficace en terme de mémoire (on ne stocke rien) et le temps de calcul supplémentaire est négligeable.

Cependant, cela nous aurait demandé beaucoup de lignes de code supplémentaires pour retrouver l'équation utilisée pour obtenir une certaine valeur. De plus, cela affecterait la clarté et la lisibilité du code.

Nous avons donc opté pour la seconde méthode même si cette dernière réduit les performances mémoire. En effet, comme expliqué précédemment, étant donné que notre programme est écrit en python, les performances ne sont pas notre objectif principal surtout que nous sommes beaucoup plus limités en temps qu'en mémoire.

Pour l'implémenter, nous avons un tuple dans chaque case de chaque matrice. Ce tuple contient :

- l'énergie de la structure optimale entre les deux indices considérés
- l'énergie de la structure optimale entre les deux indices considérés une liste M de tuples décrivant les matrices utilisées pour calculer l'énergie contenant le nom de la matrice suivi de ses indices.

Ainsi, nous pouvons facilement retracer le chemin optimal à partir de la matrice WX aux indices $i = 0$ et $j = N - 1$ (N étant la taille de la séquence).

Il suffit d'itérer à travers la liste M, puis en fonction de la matrice rencontrée, il est possible de déterminer quels nucléotides sont appariés entre eux en suivant les règles du tableau figure 6. On fait ensuite appel récursivement à la fonction de traceback avec cette matrice.

Le traceback s'arrête alors lorsque nous rencontrons une liste M vide.

3.5 Déduction de la structure secondaire

3.5.1 Format DBN

Le format "*Dot-Bracket Notation*" (notation point-parenthèse) est utilisé pour représenter la structure de l'ARN dans le package VARNA. Elle désigne les paires de bases par des paires de parenthèses et les nucléotides non appariés par des points.

```
G  G  A  G  A  U  C  G  C  A  C  U  C  C  A
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
13 _  _  _  _  _  _  _  _  _  _  _  _  0  _
Structure(s) in DBN format:
(............).
```

FIGURE 14 – Un exemple simple du format DBN représentant un hairpin loop de taille 11 [23]

Cela est un peu plus compliqué pour les pseudonœuds. En effet, nous ne pouvons pas nous contenter de parenthèses ouvrantes et fermantes.

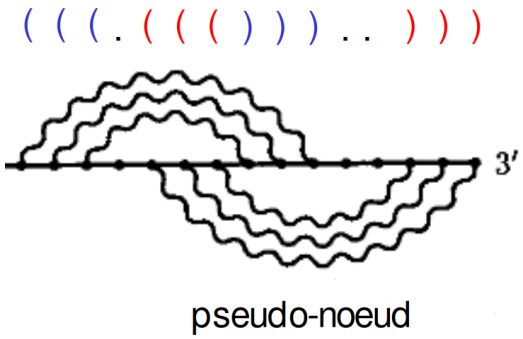


FIGURE 15 – Notation avec uniquement des parenthèses [24]

Comme on peut le voir sur cette représentation, si nous n'utilisons que des parenthèses, notre résultat serait faux. VARNA générera une structure où les nucléotides représentés par les trois parenthèses ouvrantes seraient appariés aux nucléotides représentés par les trois parenthèses fermantes.

Pour remédier à ce problème, nous avons choisi d'utiliser d'autres caractères : `[, ], {, }, < et >` supportés par le format DBN.

Si nous reprenons notre exemple précédent, avec l'introduction des nouveaux caractères, nous pouvons avoir une représentation de pseudonœud correcte :

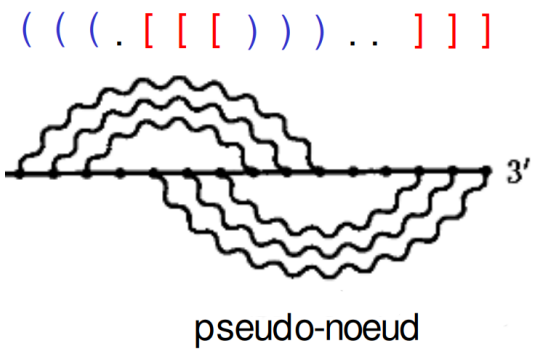


FIGURE 16 – Notation avec des parenthèses et des crochets [25]

Cet exemple reste le plus courant, mais on peut tout à fait trouver des configurations de pseudonœuds plus compliquées, comme, par exemple, compris dans une multiloop.

3.5.2 La grammaire formelle

`..(((((...)).((.((..)).)))).)).(((.[[[)]).]]]`

Voici la séquence en format DBN de la figure 12. Il est difficile de déterminer quels types de structures elle contient à partir de cette séquence.

C'est pour cette raison que nous voulions un résumé des structures contenues dans une séquence.

Notre objectif ici est donc de créer une grammaire formelle permettant de définir le langage formel des structures secondaires de l'ARN afin de l'implémenter en C à l'aide des outils flex et bison.

Notre grammaire (sans les pseudonœuds) est constituée de la manière suivante :

Les terminaux sont {`.`,`(`,`)`}, les non-terminaux sont {SEQUENCE, POINTS, PAIRE, SURFACE, MULTILOOP}et SEQUENCE est l'axiome. Les règles de production sont :

SEQUENCE → POINTS
SEQUENCE → POINTS PAIRE SEQUENCE
POINTS → ϵ
POINTS → POINTS `.`
PAIRE → `(` SURFACE `)`
SURFACE → POINTS
SURFACE → POINTS PAIRE POINTS
SURFACE → MULTILOOP
MULTILOOP → POINTS PAIRE MULTILOOP
MULTILOOP → POINTS PAIRE POINTS PAIRE POINTS

La première règle du non-terminal SURFACE correspond aux surfaces irréductibles d'ordre 1 (les hairpins) et la deuxième règle correspond à celles d'ordres 2 (stem, bulge ou internal loop que l'on différenciera à l'aide de bison).

L'ajout des pseudonœuds rend la grammaire plus compliquée d'autant plus que différentes configurations de pseudonœuds existent.

Nous avons tout de même tenu à implémenter la configuration la plus simple (topologie *a1* de la figure 4). Nous avons donc ajouté les règles suivantes :

SEQUENCE → POINTS PSEUDONOEUD
PSEUDONOEUD → PN_PAIRE POINTS CROCHETS_F
PN_PAIRE → `(` POINTS PN_PAIRE POINTS `)`
PN_PAIRE → `(` POINTS CROCHETS_O POINTS `)`
CROCHETS_O → `[`
CROCHETS_O → `[` CROCHETS_O
CROCHETS_F → `]`
CROCHETS_F → `]` CROCHETS_F

Nous ajoutons alors PSEUDONOEUD, PN_PAIRE (pour PN pour pseudonœud), CROCHETS_O (O pour ouvrant) et CROCHETS_F (F pour fermant) aux non-terminaux ainsi que `[` et `]` aux terminaux.

Les fichiers lex et yacc utilisant cette grammaire sont disponibles sur le github du projet. Le programme affiche ainsi le résultat suivant pour la séquence `..(((((...)).((.((..)).)))).)).(((.[[[)]).]]]` :

```
Secondary structure analysis of the DBN sequence:
        hairpin of size 3
        stem
INVALID hairpin of size 2
        stem
        internal loop of size 3
        stem
        multiloop with 2 structures
        right bulge of size 1
        stem
        pseudoknot
Total counts:
1 hairpin
4 stems
1 bulge
1 internal loop
1 multiloop
1 pseudoknot
```

FIGURE 17 – Résultat du programme pour la séquence citée ci-dessus [26]

Notons que la ligne “INVALID hairpin of size 2” provient du fait que nous ne possédons pas de données chiffrées dans l’article [10] concernant les hairpins de taille inférieure à trois. Nous les considérons alors comme étant invalides.

4 Difficultés liées à l’algorithme

4.1 Problème d’importation

Nous avons choisi d’écrire nos fonctions pour remplir les matrices dans des fichiers différents pour une plus grande clarté. Dans chaque fichier, nous avons donc dû importer certaines de ces fonctions. Nous avions tout d’abord choisi le format “from matrix_xx import matrix_xx”. Malheureusement, nous nous sommes confrontés à un problème d’importation circulaire.

L’importation circulaire est un problème récurrent en python notamment pour des fonctions s’appelant mutuellement.

Prenons par exemple les fonctions matrix_wx et matrix_vx respectivement dans les fichiers matrix_wx.py et matrix_vx.py. La fonction matrix_wx a besoin de la fonction matrix_vx qui a elle-même besoin de la fonction matrix_wx ce qui pose donc le problème d’importation circulaire.

En effet, nous demandons une fonction spécifique du fichier matrix_wx.py en mettant “from matrix_wx import matrix_wx” alors que ce dernier n’a pas fini d’être initialisé.

Ainsi, au lieu d’importer chaque fonction au cas par cas, il est préférable d’importer tous les fichiers en entier préalablement.

Pour régler cela, nous avons décidé d’utiliser le format “import matrix_xx”. Cela a eu pour impact d’alourdir légèrement notre code car tous nos appels aux fonctions matrices sont maintenant précédés du nom de fichier (exemple : matrix_vx.matrix_vx).

4.2 Implémentation de la grammaire formelle

Nous avions comme objectif d’offrir à l’utilisateur la possibilité d’utiliser La grammaire formelle pour directement lister les différentes structures secondaires rencontrées.

Cependant nous tenions à ce que notre programme soit utilisable sur tous les principaux systèmes d’exploitation mais les différences entre les exécutables sur Windows et ceux sur Linux ou Mac OS apportent des problèmes de portabilité.

D’autant plus que le programme demanderait alors d’être compilé avant d’être utilisé ce qui diffère encore une fois d’un système d’exploitation à un autre.

Nous avons donc pris la décision d’en faire un outil externe et optionnel.

5 Fonctionnement du programme

5.1 README

Le README expliquant tout le fonctionnement en détail du programme est disponible sur github : https://github.com/Nabil-hamoudi/Program_RNA-Structure_Python.

Vous trouverez également l'ensemble de notre code.

5.2 Ce que le programme permet

Nous avons essayé de rendre notre programme le plus simple d'utilisation possible, notamment en rédigeant un README complet et précis.
Nous allons faire un rapide descriptif de ce que le programme peut faire, avec différents flags.

Le programme principal s'exécute avec le fichier python algo.py.

5.2.1 Entrer sa séquence d'ARN

Afin de lancer le programme, l'utilisateur doit choisir une séquence d'ARN à analyser. Pour cela, il dispose du flag *-i* ou *–input* qu'il fait suivre de la séquence de son choix. La séquence peut être de l'ARN (donc composée des nucléotides A, C, G et/ou U) ou de l'ADN (dont les T seront remplacés par des U afin de devenir de l'ARN).
Voici quelques exemples d'utilisation du flag *-i* possibles :

```
python3 algo.py -i AAAUCCAAAGCGAUUUCG
python3 algo.py -i aaauccaaagcauuucg
python3 algo.py -i AAauCCAaAGcGAUUuCG
python3 algo.py -i AAATCCAAAGCATTTCG
```

Si aucune séquence n'est entrée après le flag, un message d'erreur s'affiche.

5.2.2 Ouvrir un fichier fasta

Pour lancer le programme, il peut être possible d'opter pour une seconde option en utilisant le flag *-f* ou *--file_ input*. Ce flag peut être suivi du chemin jusqu'à un fichier fasta (auquel cas ce fichier sera lu) ou non. En effet, ce flag permet d'ouvrir un fichier fasta contenant une ou des séquence(s) d'ARN (ou d'ADN) et peut éventuellement contenir des informations supplémentaires grâce au format de fichier fasta (par exemple le nom de la séquence peut être renseigné dans le header).

Si le flag *-f* ou *--file_ input* n'est suivi d'aucun argument ou si aucun flag n'est renseigné, une fenêtre tkinter s'affichera invitant l'utilisateur à sélectionner le fichier fasta contenant la séquence (ou les séquences) de son choix.

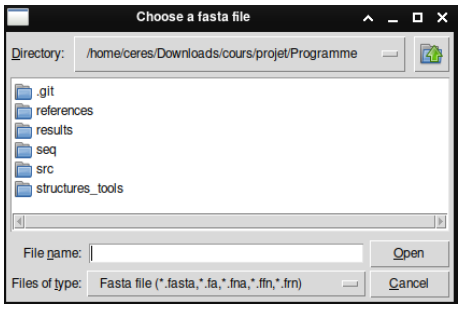


FIGURE 18 – Fenêtre tkinter permettant le choix d'un fichier fasta

5.2.3 Sauvegarder les résultats obtenus

Il est possible, en plus de l'un des deux premiers flags, de renseigner un nouveau flag *-s* ou *--save*. L'utilisateur, à nouveau, peut le faire suivre d'un chemin pour choisir l'emplacement de l'enregistrement ou ne donner aucun argument (auquel cas une fenêtre tkinter s'ouvrira pour inviter à sélectionner l'emplacement de la sauvegarde et le nom du fichier). Le nom de ce fichier est, par défaut, "*result.txt*".

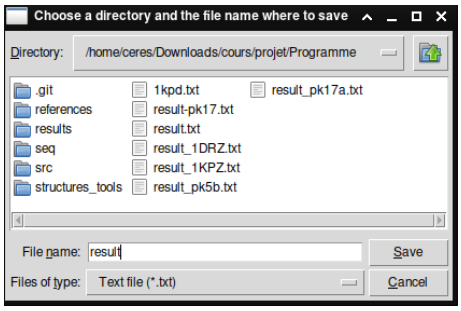


FIGURE 19 – Fenêtre tkinter permettant le choix du répertoire

Ce flag permet d'enregistrer les résultats d'une analyse d'une séquence d'ARN dans un fichier texte. Ce document comprend le nom de la séquence (si ce dernier est renseigné dans le header du fichier fasta, sinon la mention "Unknown sequence" est renseignée), l'énergie minimale de la structure, la longueur de la séquence et contient des informations sur l'appariement des nucléotides.

Le Format DBN de la séquence est également enregistré. La durée, en secondes, de l'exécution est affichée.

```
## Results for Unknown sequence ##
length of the sequence : 18 nucleotides
energy : -2.3 kcal/mol
A  A  A  U  C  C  A  A  A  G  C  G  A  U  U  U  C  G
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
15 14 13 12 11 _  _  _  _  _  _  4  3  2  1  0  _  _
Structure(s) in DBN format:
(((((.....)))))..

The algorithm took 5.53s.
```

Si le fichier fasta à l'origine de l'analyse comprend plusieurs séquences, l'ensemble des résultats pour chaque séquence est contenu dans ce même document.

5.2.4 Afficher et sauvegarder un graphe

Pour une représentation plus visuelle de l'appariement des nucléotides et de la structure secondaire, un graphe peut être créé. Pour cela, il faut utiliser le flag *-g* ou *--graph*, qui permet de sauvegarder une représentation de la structure secondaire d'une séquence.

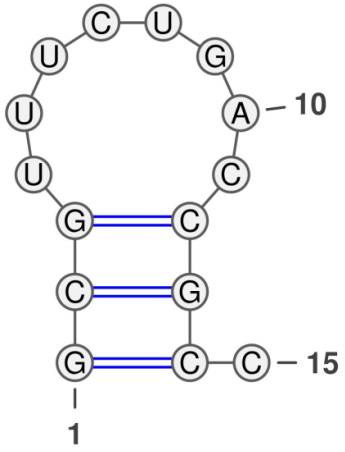


FIGURE 20 – Exemple d'un graphe généré par l'appel du flag *-g*
les numéros indiquent la position des nucléotides dans la séquence

Si, lors de l'exécution du programme principal, plusieurs séquences sont analysées, un graphe par séquence sera généré.

Ainsi, si vous analysez cinq séquences, vous obtiendrez cinq graphes à l'issue de l'exécution.

Ce flag peut être suivi d'un argument indiquant le chemin jusqu'à un répertoire où le dossier '*results*' contenant le ou les graphes sera créé. En revanche, si aucun argument n'est donné, il sera possible de choisir le dossier où sauvegarder grâce à une fenêtre tkinter.

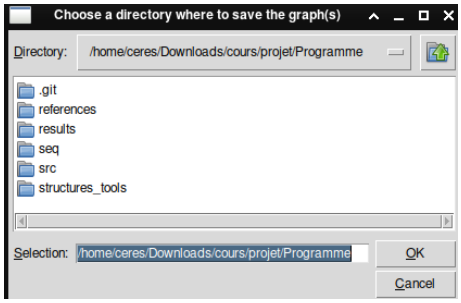


FIGURE 21 – Fenêtre tkinter permettant le choix du dossier de sauvegarde

5.2.5 Afficher le traceback

Le dernier flag possible est *-t* ou *--traceback*.

Ce flag est utilisé à des fins de débogage et nous a principalement servi pour vérifier rapidement nos résultats et repérer d'éventuels problèmes.

Il permet d'afficher le traceback, c'est-à-dire, pour chaque récursion, la matrice actuelle, les indices étudiés, le meilleur score et les matrices de la prochaine récursion.

À l'aide de ces informations, nous pouvons donc suivre l'évolution des choix de l'algorithme quant à l'appariement des nucléotides.

Voici un exemple de l'affichage des résultats contenant le détail du traceback :

```
Current matrix : wx
indices : (0, 14)
best score : -2.5
trace vx (0, 13)
-------------------------------------------------------------------------------
Current matrix : vx
indices : (0, 13)
best score : -2.1
trace EIS2 (0, 13, 1, 12)
trace vx (1, 12)
-------------------------------------------------------------------------------
Current matrix : vx
indices : (1, 12)
best score : 1.3
trace EIS2 (1, 12, 2, 11)
trace vx (2, 11)
-------------------------------------------------------------------------------
Current matrix : vx
indices : (2, 11)
best score : 3.3
trace EIS1 (2, 11)

## Results for Unknown sequence ##
length of the sequence : 15 nucleotides
energy : -2.5 kcal/mol
G  C  G  U  U  U  C  U  G  A  C  C  G  C  C
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
13 12 11 _  _  _  _  _  _  _  _  2  1  0  _
Structure(s) in DBN format:
(((........))).
The algorithm took 1.74s.
```

Notons que si les flags *-i* et *-f* ne peuvent pas être utilisés ensemble, tous les autres flags peuvent être présents ou non sans que cela ne génère d'erreur.

5.3 Reconnaître les structures

Nous avons mis à disposition un outil externe au programme avec son propre README qui permet de lister les différentes structures à partir d'une séquence DBN en utilisant La grammaire formelle.

Ce dernier peut être compilé facilement à l'aide du Makefile donné en tapant la commande `make` dans un terminal. Le Makefile utilise les outils bison, flex et gcc pour compiler le programme.

Il suffit ensuite de le lancer en donnant un fichier contenant une séquence DBN en argument avec la commande :

```
./find_structures FICHIER
```

ou en donnant directement une séquence DBN avec :

```
echo "SEQUENCE DBN" | ./find_structures
```

6 Comparaison des résultats

6.1 Performances

Dans un premier temps, nous avons décidé de tester des séquences [31]de longueur différentes afin d'observer l'évolution de temps d'exécution.

Nom de séquence	Longueur (nt)	Temps (secondes)	Énergie (kcal/mol)
1YLS_1	11	0.3286	-0.7
wyatt_PK17a	15	2.0698	-3.1
wyatt_PK5a	16	3.1827	-3.5
wyatt_PK17b	18	6.7648	-5.7
wyatt_PK5b	20	13.5081	-5.6
wyatt_PK17	23	35.541	-9.3
wyatt_PK5	26	83.4458	-7.74
1YG4	28	141.0355	-8.3
1KPD	32	374.2074	-20.09
HIVRT_I.3-2	35	701.5918	-14.55
2D1A_1	39	1570.0411	-22.6

TABLE 1 – Tableau de nos résultats [27]

On observe facilement que plus la séquence est longue, plus le temps d'exécution est long. Cela est très visuel sur un graphique.

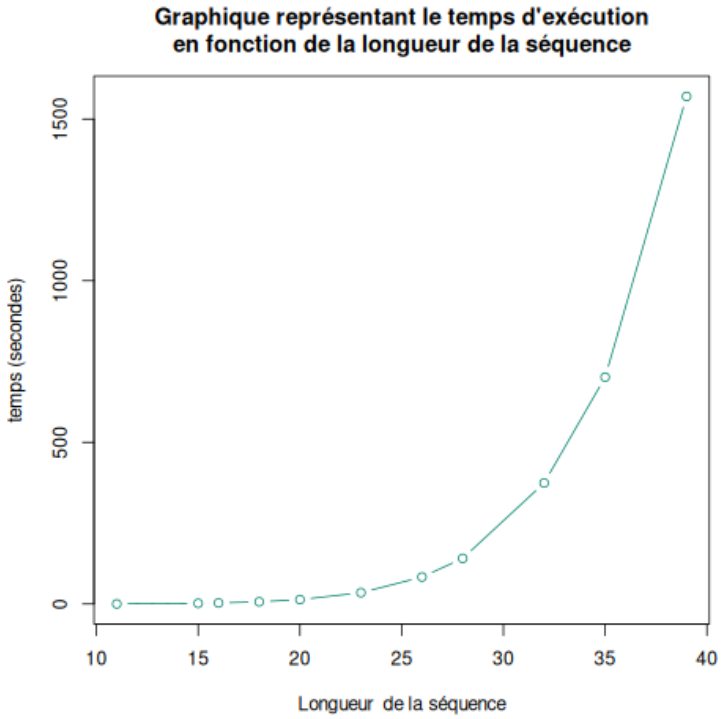


FIGURE 22 – Graphique réalisé avec nos résultats de la table 1 [28]

Pour onze nucléotides, notre programme prédit une structure en moins d'une seconde. Pour 26 nucléotides, il met environ une minute vingt. Pour 39 nucléotides, le programme met plus de vingt-six minutes à trouver une structure.

L'algorithme des auteurs est en $\mathcal{O}(N^6)$. Comme on peut le voir ici, la courbe semble bien avoir la même allure qu'une fonction polynomiale de degré 6.

6.2 L'algorithme des auteurs

Lors de nos recherches, nous avons trouvé le programme des auteurs [29] codé en C. L'idée de comparer nos résultats avec les leurs nous a paru être une bonne idée afin de vérifier la qualité de notre programme.

	Résultats programme article		**Nos résultats**		
Nom ou n°accession	ΔG (en kcal/mol)	structures	ΔG (en kcal/mol)	structures	commentaires
1YLS_1	-1.00	hairpin	-0.7	hairpin	même structure, énergie différente
wyatt_PK17a	-1.90	hairpin	-3.1	hairpin	même structure, énergie différente
wyatt_PK5a	-2.9	hairpin	-3.5	hairpin	même structure, énergie différente
wyatt_PK17b	-3.90	hairpin	-5.7	hairpin	même structure, énergie différente
wyatt_PK5b	-3.70	hairpin	-5.6	hairpin	même structure, énergie différente
wyatt_PK17	-6.29	pseudoknot	-9.3	pseudoknot	même structure, énergie différente
wyatt_PK5	-6.11	pseudoknot	-7.74	pseudoknot	même structure, énergie différente
1YG4	-5.04	pseudoknot	-8.3	pseudoknot	même structure, énergie différente
1KPD	-14.74	pseudoknot	-20.09	pseudoknot	même structure, énergie différente
HIVRT_I.3-2	-10.86	pseudoknot	-14.55	pseudoknot	même structure, énergie différente
2D1A	-17.00	pseudoknot	-22.6	internal loop et hairpin	structure différente, énergie différente

Hormis la dernière séquence, pour laquelle nous n'obtenons pas la même structure, nos prédictions sont identiques à celles des auteurs.

En revanche, nos énergies diffèrent un peu. Notre programme donne, la plupart du temps, une énergie légèrement plus faible que celle de Rivas et Eddy. Cependant, nous restons toujours proches, les différences d'énergie ne semblent pas aberrantes.

Ces variations peuvent être dûes à des différences dans les paramètres. Etant donné que l'on a trouvé les nôtres dans l'article *"Improved free-energy parameters for predictions of RNA duplex stability"* [10] et sur le site *"Nearest Neighbors DB"* [30], on ne peut être certains que les auteurs les aient utilisés également.

Nous venons de comparer les séquences pour lesquelles nous avions observé le temps d'exécution dans la partie Performances. Etant donné qu'elles représentent presque toutes soit un hairpin, soit un pseudonœud, nous avons analysé quelques séquences supplémentaires afin d'avoir une plus grande diversité de structures.

	Résultats programme article		**Nos résultats**		
Nom ou n°accession	ΔG (en kcal/mol)	structures	ΔG (en kcal/mol)	structures	commentaires
1BVJ	-7.9	hairpin et bulge	-10.0	hairpin et bulge	mêmes structures, énergie différente
2VAL	-23.7	hairpin	-24.2	hairpin	même structure, énergie différente. Après vérification, nous aurions dû trouver un pseudonœud
1BIV	-12.10	hairpin et 2 bulges	-14.9	hairpin et internal loop	structures et énergies différentes. Après vérification, nous avons la structure attendue
2AB4	-7.8	hairpin	-10.7	hairpin	même structure, énergie différente
1G70	-10.7	hairpin, bulge et internal loop	-13.5	hairpin, bulge et internal loop	mêmes structures, énergie différente

Avec ces cinq séquences supplémentaires, nous confirmons la tendance : notre algorithme prédit fréquemment la même structure que le programme des auteurs. Ceci est assez positif puisque cela était notre objectif.

On peut néanmoins faire une critique : le programme n'est pas infaillible. Il se trompe parfois, comme on le voit avec la séquence *2VAL*. En effet, notre programme et celui des auteurs échouent à prédire le pseudonœud.

7 Conclusion

7.1 Bilan

Nous avons vu dans la partie Comparaison des résultats que notre objectif principal (celui de reproduire le programme des auteurs à partir de leur article) est une réussite.

Bien entendu, il n'est pas identique : les énergies diffèrent, les structures parfois également. Sur 16 séquences testées et présentées dans ce rapport, notre algorithme a renvoyé 14 fois la (ou les) même(s) structure(s) que celui des auteurs. Cela représente 87.5% de réussite, ce qui n'est pas négligeable.

Ce que l'on peut donc dire c'est que le programme permet une prédiction de pseudonœuds limitée (seuls certains pseudonœuds et il ne les trouve pas systématiquement) mais fonctionne pour un grand nombre de séquences.

7.2 Mélange de connaissances

Ce projet est celui de notre fin de licence et reflète bien que l'ensemble de nos connaissances et acquis sont utiles.

Nous avons eu besoin de nos cours d'algorithmique, de programmation, de biologie, de compilation et d'anglais principalement. Bien entendu, ce projet nous a forcé à faire beaucoup de recherches, à utiliser maintes et maintes fois la documentation python. Cela a été très agréable de faire un programme qui réunit tous ces points, et pas seulement une matière spécifique.

De plus, la liberté que nous avions quant à nos choix d'implémentation, de rédaction était très appréciable.

7.3 La compréhension

Le plus compliqué a été, sans aucun doute, de comprendre l'article, les récursions, les différentes structures, comment lier tout cela. Pour quelques lignes de code, finalement plutôt simples, il fallait parfois une ou deux heures (voire plus dans certains cas) de recherche et de discussion pour comprendre exactement comment cela fonctionnait et quel résultat nous attendions.

Le fait de réaliser un projet à plusieurs a également été d'une grande aide pour la compréhension. Lorsqu'un sujet n'est pas compris par certains, il l'est certainement par d'autres. Nous avons pu répartir nos recherches et s'expliquer les uns aux autres ce que nous avions trouvé.

Devoir l'expliquer avec nos mots et expliquer comment on envisage le code pour cela permet également de vérifier que nous avons bien compris. Nous trouvons important que chacun ait une compréhension globale du projet, et pas simplement de la partie que l'on a codée soi-même.

7.4 Le mot de la fin

Nous ne regrettons pas notre choix d'article. En effet, nous avons déjà travaillé sur de la programmation dynamique, chose pour laquelle nous n'étions pas forcément à l'aise.

Nous avons dû faire beaucoup de recherches bibliographiques, s'informer sur un sujet dont nous ignorions beaucoup de choses. Cela nous a permis de découvrir ou d'approfondir un aspect de la biologie et surtout de la bio-informatique qui nous a intéressé.

Ce projet a également été un challenge en termes de gestion du temps. Parce que ce dernier est conséquent et que nous ne voulions pas "bâcler" le travail, nous avons eu une organisation très précise tout en restant relativement flexibles pour s'adapter aux imprévus.

A l'issue de ce projet, nous avons tous la sensation d'avoir progressé, aussi bien en programmation qu'en méthodologie. Cela fut très enrichissant.

8 Planning

Voici le détail de notre planning, aidant à l'organisation de notre travail et à finir dans les temps.

- **10 Mars** : choix du sujet, validation du groupe par S.VIAL
- **10 au 17 Mars** : lecture approfondie de l'article, annotations, liste de questions à poser
- **17 Mars** : réunion avec S.VIAL
- **20 au 25 Mars** : chacun de son côté, réfléchir à l'algorithme, aux matrices, à leur remplissage
- **25 Mars** : mise en commun des idées, réunion pour répartition des premières tâches (date limite 1er Avril) répartition suivante :
  - ✓ Création, initialisation des matrices et vhx (Chloé)
  - ✓ Fonction pour whx (Marwane)
  - ✓ Fonctions pour zhx et yhx (Nabil)
  - ✓ Fonctions pour wxi, vx et wx (Julie)
- **31 Mars** : parametres.py pour simplification (dictionnaire, fonctions coaxial stacking, dangle etc...)
- **1er avril** : bilan sur les premières matrices, discussion autour de la suite et plus spécifiquement, sur les surfaces irréductibles, recherche d'articles les concernant (durée limite : 4 jours).
- **4 Avril** : discussion sur les surfaces irréductibles, lecture d'articles/théorèmes les mentionnant.
- **6 Avril** : attribution de nouvelles tâches de code (date limite 10 Avril) établissement du plan provisoire du rapport et début de la rédaction répartition suivante :
  - ✓ Dangle équations : Julie
  - ✓ EIS1 : Nabil
  - ✓ EIS2 : Chloé et Marwane
- **13 avril** : ajout des équations de récurrence incluant les dangles et le coaxial stacking.
- **16 avril** : 1er test de remplissage de vx → problèmes
  - ✓ limite récursion atteinte
  - ✓ syntaxe (oublis symboles...)
  - ✓ problèmes d'indices
  - ✓ problème d'appel récursifs (la matrice s'appelle elle-même)
  - ✓ impossibilté d'utiliser des variables globales (matrices, séquence)
  - ✓ changements de toutes les matrices, parameters, les fonctions, create matrices (ajout wxi notamment)
- **17 avril** : encore quelques changements, débogage, 1er test réussi pour vx ; test réussi pour toutes les autres matrices
- **19 avril** réunion avec S.VIAL pour discuter de l'état de l'art et du traceback
- **21 avril** : réalisation du traceback
- **22 avril** : correction et modification du traceback pour inclure les pseudo noeuds + test de séquences + premiers affichages
- **23 avril** : attribution de nouvelles tâches
  - ✓ Julie : lecture fichier FASTA + vérifier ARN + recherche de fichiers tests
  - ✓ Nabil : gestion ligne de commande et sauvegarde des résultats
  - ✓ Chloé et Marwane : affichage graphique + correction traceback + affichage résultats
- **24 Avril** : réunion BU pour derniers points sur affichage, format des fichiers contenant les séquences, organisation de la prochaine semaine (avant l'arrivée des vacances) (18 points abordés)
- **27 Avril** : grande partie interaction algo-utilisateur finie, fonction pour graphe finie, fonction pour récupérer structures secondaires en cours
- **28 Avril** : Relecture, amélioration des fonctions (sous-fonctions), harmonie du code, rédaction d'une première grande partie du rapport
- **29 Avril** : test de détails concernant le parser et la lecture de fichiers fasta, suggestions, modifications et gestion d'erreurs
- **30 Avril** : dernières modifications mineures, rédaction des derniers commentaires et docstring

- **1 Mai** : rédaction du rapport durant les prochains jours
- **3 Mai** : ajout de la résolution concernant les graphes et prise en charge des espaces dans les noms de fichiers
- **4 Mai** : ajout de la partie grammaire formelle, derniers paragraphes du rapport
- **5 Mai** : test du programme sur différentes tailles de séquences, graphiques de résultat, ajout des références dans le rapport
- **6 Mai** : mise à jour du rapport après corrections, ajouts et ré arrangement de certaines parties, corrections mineures du code
- **7 Mai** :
  - ✓ modification imprévue du code, nouveaux calculs de tous les résultats avec la nouvelle version
  - ✓ réunion pour oral : établissement du plan, répartition des rôles, réalisation du diaporama (beamer, ovreleaf)
- **8 Mai** : Fin du rapport, fin du diaporama

9 Références

[1] T. Jiang, G. Lin, B. Ma, and K. Zhang, “A general edit distance between rna structures,” 2022.

[2] R. NUSSINOV and A. B. JACOBSON, “Fast algorithm for predicting the secondary structure of single-stranded rna,” 1980.

[3] E. Rivas and S. R. Eddy, “A dynamic programming algorithm for rna structure prediction including pseudoknots,” 1999.

[4] Christian Schudoma, https://www.researchgate.net/figure/Canonical-Watson-Crick-Base-Pairs-Basepairings-via-hydrogen-bonds-occur-between-a_fig2_228958747

[5] M. K. Gupta, https://www.researchgate.net/figure/Different-types-of-RNA-Secondary-Structures-1-First-structure-represents-a-stem-withfig1261030242

[6] Page wikipédia : Pseudonoeuds, https://fr.wikipedia.org/wiki/Pseudon%C5%93ud#/media/Fichier:PseudoknotFold.svg

[7] Michaël Bon. Prediction de structures secondaires d’ARN avec pseudonœuds. Life Sciences [q-bio]. Ecole Polytechnique X, 2009. English. NNT : . pastel-00005806

[8] Dessins réalises sur Xournal++ par GODET Chloé et GROSJACQUES Marwane

[9] Diagrammes de Feynman, figure 6 "recursion for vx truncated at $\mathcal{O}(2)$" page 5 de l’article [3].

[10] S.M. FREIER, R.KIERZEK, J.A.JAEGER, N.SUGIMOTO, M.H. CARUTHERS,T.NEILSON and D.H.TURNER, "Improved free-energy parameters for predictions of RNA duplex stability", Decembre 1986, https://www.pnas.org/doi/epdf/10.1073/pnas.83.24.9373

[11] E.Rivas and S.R.Eddy, "Complete set of recursions dor the pseudknot algorithm", supplementary materiel for [3], http://eddylab.org/publications/RivasEddy99/suppmaterial_JMB.pdf

[12] VARNA : Visualization Applet for RNA. A Java lightweight component and applet for drawing the RNA secondary structure http://varna.lri.fr/

[13] Traduction en français du tableau de l’article [11], page 2

[14] Dessin réalisé sur XOURNAL++ par GODET Chloé

[15] Tableau de paramètres disponible dans l’article [3] page 7.

[16] Tableau de paramètres disponible dans l’article [3] page 10.

[17] Deux diagrammes de Feynman représentant des dangles, figure 2 de l’article [11]

[18] Trois diagrammes de Feynman représentant des dangles, figure 14 de l’article [11]

[19] Schéma d’un coaxial stacking flush sur les paires terminales AU et CG, /urlhttps ://rna.urmc.rochester.edu/NNDB/turn example-1.html

[20] Schéma d’un coaxial stacking mismatch-mediated sur les paires terminales GG et CG, https://rna.urmc.rochester.edu/NNDB/turner04/coax-example-2.html

[21] Figure 2 : Diagrammatic representation of the most relevant RNA secondary structures, including a pseudoknot de l’article [3]

[22] Schémas représentant les deux méthodes citées, réalisés par GROSJACQUES Marwane

[23] Capture d’écran de l’affichage de notre programme

[24] Capture d’écran de la [21] avec ajout de la notation BDN correspondante grâce à Xournal++ et traduction de la légende en français

[25] Capture d'écran de la [21] avec ajout de la notation BDN correspondante grâce à Xournal++ et traduction de la légende en français

[26] Capture d'écran du résultat du programme (sur nos ordinateurs) pour la séquence :
`..(((((...)).((.((..)).)))).)).(((.[[[)]).]]]`

[27] L'ordinateur utilisé pour obtenir ces résultats est toujours le même : HP ENVY x360 Convertible 13-ay1x, 16GO RAM, AMD RYZEN 7 5800U, Arch Linux x86_64
Séquences trouvées sur le site : https://www.rcsb.org/

[28] Graphique réalisé en R, sauvegardé en png

[29] Programme disponible sur github https://github.com/EddyRivasLab/PKNOTS

[30] site contenant des paramètres importants dont nous nous servons, https://rna.urmc.rochester.edu/NNDB/turner04/

[31] Toutes les séquences trouvées et testées sont disponibles sur les sites https://www.rcsb.org/ et http://www.rnasoft.ca/strand/

[32] deux autres sites nous ayant permis de comprendre notre article lors de nos recherches : https://albuquerque.bioinformatics.uottawa.ca/Papers/JournalPublication/1985_Sankoff.pdf, https://albuquerque.bioinformatics.uottawa.ca/Papers/JournalPublication/1984_Zuker_Sankoff.pdf