Parsing avec erreurs par un protomate

Jessie Mahé

MAHÉ Jessie

Stage 4EII :
Parsing avec erreurs par un protomate

Encadrant IRISA : François COSTE
Encadrant INSA : Laurent BEDAT

Année 2006/2007

Remerciements

Je tiens à remercier Jacques Nicolas, responsable de l'équipe Symbiose, pour m'avoir permis d'effectuer mon stage dans les meilleures conditions. Je remercie aussi François Coste, chercheur, pour m'avoir encadrée et pour m'avoir laissé la marge de manœuvre qui a rendu ce stage si intéressant. Je remercie également Goulven Kerbellec, doctorant, pour ses conseils avisés ainsi que sa patience. Enfin, je remercie Mathieu Giraud et Laetitia Guillot avec qui j'ai beaucoup apprécié travailler, ainsi que toute l'équipe symbiose pour son accueil chaleureux.

Table des matières

Résumé du stage

Mon stage a consisté en l'implémentation en langage C d'un parseur permettant de scanner rapidement une banque de données de séquences protéiques. Le scan permet de connaître l'appartenance des séquences à une famille modélisée par un automate pondéré. Les séquences rendues par le scan peuvent appartenir exactement à la famille ou être à une faible distance de celle-ci. À la fin du scan on obtient un score résumant l'adéquation automate-séquence. Ce programme a été complété par une option permettant de mémoriser pour chaque séquence de la banque de données quel chemin a été pris dans l'automate, et l'évolution du score en fonction de l'avancement dans la séquence. Nous avons également réfléchi à la mise en ligne du programme et pour cela nous avons travaillé sur le découpage en modules de l'interface Web.

Mots clés

Bio-informatique
Automate pondéré
Parsing avec erreur
Algorithme de Viterbi
Parsing

1. Introduction

1.1. Présentation de l'IRISA et de l'équipe Symbiose

En 1968, l'Université de Rennes 1 et l'INSA de Rennes se sont associés pour créer sur le campus universitaire de Beaulieu, dans leurs établissements respectifs, des filières d'enseignement supérieur en informatique. En même temps se met en place un laboratoire de recherche en systèmes et langages. En 1975, le laboratoire est reconnu comme laboratoire associé au CNRS et prend le nom d'Institut de Recherche en Informatique et Systèmes Aléatoires (Irisa). Actuellement, l'Irisa regroupe environ 530 personnes dont plus de 200 chercheurs ou enseignants chercheurs, 175 chercheurs en thèse, et 90 ingénieurs, techniciens, administratifs. Sous la direction de Patrick Bouthemy, les recherches de l'Irisa se développent dans plus de 25 équipes, dont l'équipe Symbiose.

Ce groupe de recherche, au sein duquel mon stage s'est déroulé, est dirigé par Jacques Nicolas. Symbiose est un projet de recherche en bio-informatique, qui s'intéresse d'une manière générale à la modélisation des données génomiques , dans le but d'assister le biologiste moléculaire.

1.2. Problématique

Au sein de l'équipe Symbiose, un outil d'apprentissage automatique d'automates du nom de Protomata Learner [1] a été développé. Il sert à la caractérisation et la modélisation de familles structurelles ou fonctionnelles de protéines. L'automate généré peut être utilisé pour rechercher dans les banques de données de séquences protéiques de nouvelles séquences pouvant appartenir à la famille qu'il modélise. Il était donc intéressant d'implémenter un programme réalisant cette opération.

Le but de ce stage a donc été de réaliser et de mettre à disposition sur le Web un programme capable de scanner rapidement une banque de séquences protéiques pour identifier les protéines appartenant au langage représenté par un automate ou à faible distance de celui-ci.

Dans un premier temps je détaillerai l'étude préliminaire comprenant quelques bases théoriques ainsi que l'étude de l'existant, et en deuxième partie j'expliquerai le déroulement de mon stage, c'est à dire la démarche que j'ai entreprise pour réaliser ce programme, la réalisation du programme et les résultats que j'ai obtenus.

2. Étude préliminaire

Au début de mon stage je devais choisir la manière dont je souhaitais réaliser le parsing avec erreurs par un protomate. La première étape a donc été de me documenter, de répertorier ce qui existait et de voir quels outils pouvaient m'être utiles. J'ai également appris des notions théoriques concernant la biologie. Un algorithme appelé « algorithme de Viterbi » était une piste pour le futur programme, je l'ai donc étudié. Un outil du nom de Wapam [2] avait été également mentionné mais comme son concepteur n'était plus dans l'équipe Symbiose, il fallait l'étudier afin de savoir ce qu'il permettait de faire exactement et comment il le faisait. Ainsi je saurai si je pouvais le modifier et l'adapter au problème.

2.1. Bases de biologie

J'ai commencé par apprendre un certain nombre de termes liés à la bio-informatique. En effet les banques de données sur lesquelles j'ai travaillé contiennent des séquences de protéines et le programme que l'on souhaite réaliser traite ces séquences. Il a également plusieurs options qui demandent quelques connaissances en biologie.

La bio-informatique étant la spécialité de l'équipe dans laquelle j'ai travaillé, la documentation a été assez facile, même si le vocabulaire à retenir était très varié et relativement dense. Voici donc quelques termes utiles à la compréhension de ce rapport :

- Une protéine est une séquence d'acides aminés, un acide aminé étant codé dans l'ADN par trois acides nucléiques. Il existe 20 acides aminés présents dans le code génétique, dont les symboles sont les suivants :
  A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, et V.

- Quand on parle d'une base protéique cela signifie que l'on travaille avec des séquences d'acides aminés, une base nucléique signifie que l'on manipule des séquences d'acides nucléiques. Le génome est l'ensemble du matériel génétique d'un individu ou d'une espèce, il est composé de gènes formés d'acides nucléiques.

- Les acides nucléiques sont au nombre de 4 (A, C, G ou T) et sont complémentaires entre eux. Ainsi, l'adénine A est complémentaire à la thymine T et la guanine G est complémentaire à la cytosine C. Le brin complémentaire d'une séquence d'acides aminés correspond à inverser le début et la fin de la séquence et à complémenter tous les acides nucléiques composant la séquence.

- La substitution d'un acide aminé se traduit par le changement de cet acide aminé par un autre.

2.2. Automate pondéré ou WA (Weighted Automata)

Après les notions de biologie, j'ai lu la thèse de Mathieu Giraud [4] traitant des automates pondérés et de la recherche de motif. J'ai également lu de nombreux articles sur les automates, en effet ce sujet est très bien documenté.

2.2.1. Définition

Soit Z l'ensemble des entiers, un automate pondéré est un quintuplet (S, E, T, D, Q) où:

- S est l'ensemble des états constituant cet automate,
- E est un alphabet,
- T est inclus dans SxExSxZ et est l'ensemble des transitions. Une transition est donc caractérisée par un quadruplet (B, F, L, P), où B est l'état de départ, F l'état d'arrivée, L un élément de l'alphabet caractérisant la condition de transition et P la pondération de cette transition.
- D est inclus dans S et est l'ensemble des états initiaux de l'automate,
- Q est inclus dans S et est l'ensemble des états finaux.

L'automate prend en entrée une séquence d'éléments de l'alphabet, qu'il va accepter ou rejeter. Dans le cas de la reconnaissance de la séquence un score lui est affecté. Quand on utilise un automate pour reconnaître une séquence, on réalise ce qu'on appelle un parsing.

Au départ tous les états initiaux sont actifs. Au fur et à mesure de l'avancement dans la séquence et en fonction de l'ensemble des transitions, de nouveaux états deviennent actifs. Si à la fin de la séquence au moins un état final est actif, alors la séquence est acceptée. Dans ce cas, un score lui est attribué correspondant à la somme des poids des transitions empruntées dans l'automate.

2.2.2. Applications

Le parsing que l'on souhaite réaliser accepte les erreurs, c'est à dire que la séquence peut ne pas correspondre exactement à l'automate. Les erreurs que nous acceptons sont des substitutions d'acides aminés. Pour gérer ces erreurs le principe est le suivant : lorsqu'une transition existe entre deux états, on accepte tous les acides aminés mais avec des coûts différents. Pour les acides aminés qui correspondent exactement à l'automate on aura un coût de 0 et pour les autres on aura un coût non nul correspondant à la pénalité d'une substitution.

La première application des automates pondérés qui va nous intéresser, est le parsing à l'aide d'un protomate. Je vais donc décrire ce qu'est un protomate et donner un exemple. Pour me documenter

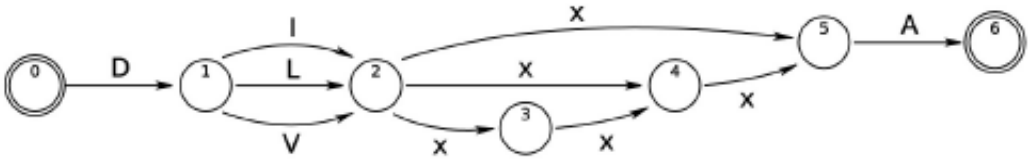
sur les protomates j'ai étudié des articles internes à l'équipe car le terme protomate est lié à un projet de l'équipe Symbiose, et il est donc difficile de trouver des documents extérieurs.

Un protomate est un automate modélisant une famille de séquences d'acides aminés, il a une notion de début et de fin par rapport à la séquence : c'est à dire que l'état initial de l'automate doit correspondre au début de la séquence et l'état final à sa fin.

Exemple :

Considérons les séquences d'acides aminés suivantes:
Séquence 1 : D-V-I-I-A
Séquence 2 : D-V-I-I-I-A-D
Séquence 3 : D-A-I-A

Considérons l'automate suivant:



Les transitions marquées d'un «x» correspondent à des transitions où tout acide aminé est accepté, une telle transition est appelée « joker ». Une suite de plusieurs jokers consécutifs est un « gap ».

La séquence 1 correspond exactement à l'automate.
La séquence 2 ne correspond pas car elle est trop longue : la fin de la séquence ne correspond pas à l'état final de l'automate.
La séquence 3 correspond à l'automate avec une erreur (substitution du 2ème acide aminé par l'acide aminé A).

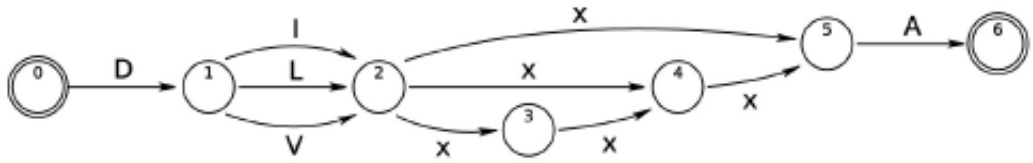
La deuxième application des automates pondérés est la recherche de motif, en effet en biologie on est souvent amené à trouver un motif dans une séquence, pour cela on peut se servir d'un automate modélisant un motif. Contrairement à précédemment avec les protomates, la notion de début et de fin n'est pas importante.

Exemple :

Soient les séquences d'acides aminés suivantes :
Séquence 1 : A-A-A-D-V-I-I-A-A-A-D-V-I-I-A -A

Séquence 2 : D-V-I-I-I-A-D
Séquence 3 : D-A-I-A
Séquence 4: D-V-A

Soit l'automate suivant :



La séquence 1 correspond exactement à l'automate : A-A-A-**D-V-I-I-A**-A-A-A-**D-V-I-I-A** -A
(on trouve deux correspondances).
La séquence 2 correspond exactement à l'automate également : **D-V-I-I-I-A**-D (une seule correspondance).
La séquence 3 correspond à l'automate avec une erreur (substitution du 2ème acide aminé par l'acide aminé A) : **D-A-I-A**.
La séquence 4 ne correspond pas à l'automate, la séquence est trop courte.

2.3. Utilisation de l'algorithme de Viterbi

Par la suite, j'ai étudié l'algorithme de Viterbi ainsi que la façon dont je pouvais l'appliquer pour mon programme. Se documenter sur l'algorithme a été plutôt facile car c'est un algorithme très connu et très utilisé, par contre je n'ai pas trouvé d'application similaire à mon problème et j'ai donc étudié cela avec mon responsable de stage.

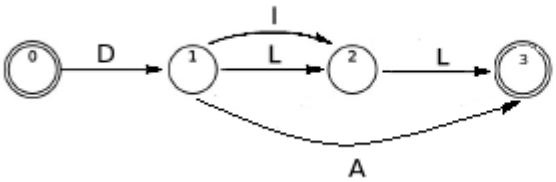
L'algorithme de Viterbi est un algorithme de programmation dynamique. Tout d'abord définissons le principe de la programmation dynamique : cela consiste à déduire la solution optimale d'un problème à partir de la solution optimale d'un sous problème. Cet algorithme permet à partir d'une séquence et d'un automate pondéré de trouver le chemin optimal, c'est à dire correspondant au meilleur score en fonction des possibles erreurs (on rappelle que le score est la somme des poids des transitions empruntées dans l'automate). Détaillons à présent l'algorithme de Viterbi appliqué à notre problème.

Soit *o* la séquence à parser à l'aide d'un automate. Pour trouver la meilleure correspondance « automate-séquence », on cherche, à partir de *o* quelle combinaison a le meilleur score. La méthode consiste à tester, pour chaque lettre de *o* les transitions qui nous donnent le coût optimal. De manière

plus générale, l'algorithme de Viterbi est utilisé dans les cas où le problème est modélisable comme une chaîne de Markov : c'est à dire que la prédiction du futur à partir du présent ne nécessite pas la connaissance du passé. Pour notre application on peut reformuler le problème ainsi : étant donné la séquence *o*, on doit trouver la séquence d'états *x* pour laquelle le coût final est optimal. Pour plus d'information sur l'algorithme de Viterbi général voir [3].

Appliquons l'algorithme de Viterbi dans un cas concret, voici un exemple de cet algorithme pour une recherche de motifs :

Soit l'automate suivant :



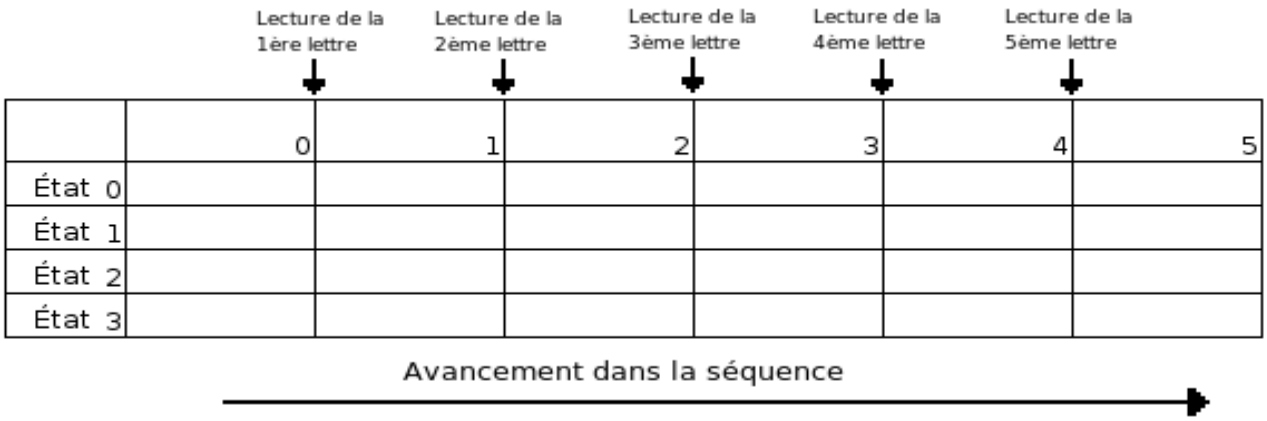
Comme expliqué précédemment, pour gérer les substitutions on accepte tous les acides aminés pour chaque transition mais avec un coût, ici on a représenté les acides aminés ayant un coût nul. Voici la matrice de coûts correspondante :

Lettres: /Transitions:	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
0->1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1->2	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1
2->3	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
1->3	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

On décide dans ce cas d'attribuer une pénalité de 1 quand l'acide aminé ne correspond pas et on se place dans le cas où l'on cherche à minimiser le coût.

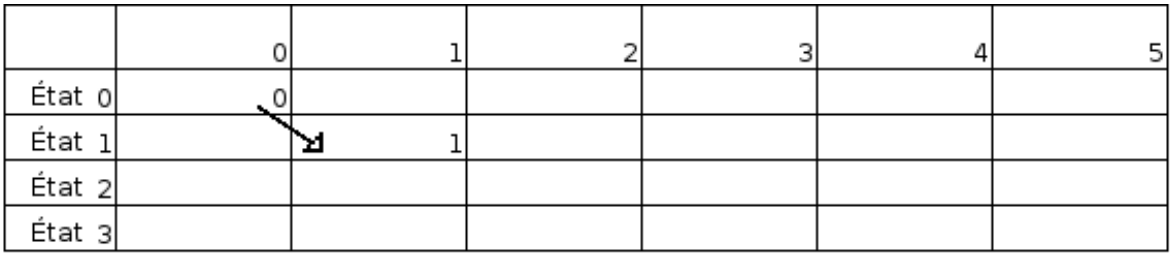
On considérera la séquence suivante : G-A-D-L-L. L'algorithme consiste à remplir un tableau dont la taille est égale à la longueur de la séquence plus un multipliée par le nombre d'états. On peut noter que dans la plupart des cas toutes les cases du tableau ne sont pas remplies, en effet les états ne sont pas tous atteints à chaque itération.

Voici le tableau correspondant à notre exemple :



Les flèches dans les tableaux ci-dessous représentent les transitions possibles en fonction de l'avancement dans la séquence. Sachant que dans une recherche de motif, le motif peut être au milieu de la séquence on doit pouvoir commencer à n'importe quel endroit de la séquence, ce qui explique que pour chaque lettre lue on a la possibilité de partir de l'état initial. Dans notre cas on part toujours avec un coût initial nul.

1ère étape : lecture de la première lettre G



On ne peut atteindre qu'un état : l'état 1 et on a un coût de 1 car la lettre G ne correspond pas à la transition 0 -> 1.

2ème étape : lecture de la deuxième lettre A

	0	1	2	3	4	5
État 0	0	0				
État 1		1	1			
État 2			2			
État 3			1			

Comme on est dans le cas de la recherche de motif à chaque lettre de la séquence on peut repartir de l'état initial, ce qui explique la flèche de l'état 0 à l'état 1. A cette étape on a donc trois possibilités de placement dans l'automate : soit on a atteint l'état final 3 avec le motif **G-A**-D-L-L et un coût égal à 1, ou on peut être à l'état 1 avec un coût de 1 ou 3ème possibilité on est à l'état 2 avec un coût de 2. Comme l'état final est atteint, on a donc un premier résultat avec un coût de 1.

3ème étape : lecture de la 3ème lettre D

	0	1	2	3	4	5
État 0	0	0	0			
État 1		1	1	0		
État 2			2	2		
État 3			1	2		

Quand 2 transitions arrivent à un même état on garde le coût le plus avantageux, ainsi ici on avait deux chemins possibles pour arriver à l'état 3, le chemin « état 0/ état 1/ état 2/ état 3 » de coût 3 (motif : **G-A-D**-L-L) ou le chemin « état 0/ état 1/ état 3 » de coût 2 (motif : G-**A-D**-L-L) et donc on ne conserve que le chemin de coût minimal. On a donc un deuxième résultat avec un coût de 2.

4ème étape : lecture de la 4ème lettre L

	0	1	2	3	4	5
État 0	0	0	0	0		
État 1		1	1	0	1	
État 2			2	2	0	
État 3			1	2	1	

On se trouve dans le même cas que précédemment : on a le chemin « état 0/ état 1/ état 2/ état 3 » de coût 2 (motif : G-**A-D-L**-L) ou le chemin « état 0/ état 1/ état 3 » de coût 1 (motif : G-A**-D-L**-L), on garde donc la deuxième possibilité.

5ème étape : lecture de la 5ème lettre L

	0	1	2	3	4	5
État 0	0	0	0	0	0	
État 1		1	1	0	1	1
État 2			2	2	0	1
État 3			1	2	1	0

Dans la dernière étape on a un coût de 0 par le chemin « état 0/ état 1/ état 2/ état 3 » et c'est le plus petit coût qu'on peut obtenir avec cette séquence. On l'obtient avec le motif DLL : G-A-**D-L-L**.

2.4. Étude du programme Wapam

Après la lecture de la thèse de Mathieu Giraud, j'ai commencé à m'intéresser à Wapam, un programme qu'il avait implémenté. Étudier Wapam m'a pris beaucoup de temps car il existait seulement une documentation utilisateur, et aucune documentation technique. De plus le code manquait de commentaires, et Mathieu Giraud n'était pas joignable. J'ai donc passé un certain temps à reprendre le code et à le commenter afin de pouvoir comprendre ce qu'il faisait exactement et comment il le faisait. Voici un aperçu du fonctionnement du programme.

Wapam peut rechercher des motifs protéiques ou nucléiques, avec ou sans erreur(s), dans des génomes ou dans des banques de données. Wapam recherche des motifs exprimés en automates pondérés. On lit chaque lettre de la séquence et pour chaque lettre on regarde les possibilités qui s'offrent à nous au niveau de l'automate : à chaque position, le poids de sa transition est additionné au score. Ce poids reflète l'adéquation d'une partie de la séquence avec la lettre lue à cette position dans

l'automate. Comme on accepte de rencontrer des motifs contenant des erreurs, on a la possibilité de fixer un seuil afin de pouvoir déterminer combien d'erreurs on décide de tolérer. Le motif est reconnu lorsque l'un des états finaux de l'automate est actif avec un score compatible avec le seuil d'erreur fixé.

Pour trouver le coût optimal lors du parsing, on doit donc remplir un tableau avec l'algorithme de Viterbi comme on l'a vu précédemment. Cela dit afin de minimiser les ressources mémoire nécessaires dans Wapam, on décide de ne pas stocker les coûts dans un tableau dont la taille t est :
t = (nombre_d_états)*(longueur_de_la _séquence +1),
mais dans un tableau de taille t' :
t' = (nombre_d_états)*2.

En effet le remplissage d'une colonne se fait uniquement à partir de son prédécesseur (car le problème est modélisable par des chaînes de Markov) et du coup on a besoin de stocker uniquement deux colonnes : la colonne courante C_i et celle qu'on doit calculer C_{i+1}. Quand on a fini de calculer la nouvelle colonne C_{i+1}, on peut effacer la colonne courante C_i , pour permettre une nouvelle itération et calculer C_{i+2} dans l'espace ainsi libéré.

Wapam compte plusieurs options telles que le parsing de séquences de base nucléique à l'aide d'un automate en base protéique, ou encore le parsing des brins complémentaires des séquences d'une base de données.

Voici les fichiers principaux du code de Wapam ainsi qu'une description de leurs fonctionnalités:

- **wa.c** : Calcul principal pour le parsing. C'est dans ce fichier qu'est codé la fonction correspondant à la boucle principale, elle utilise l'algorithme de Viterbi détaillé précédemment et elle s'effectue sur le tableau contenant toutes les transitions. Cette fonction prend en paramètre une lettre (un élement d'une séquence), un automate pondéré et un entier correspondant à la position dans le fichier de séquences. Elle retourne le coût du parsing. Ce fichier contient également des fonctions d'initialisation.
- **parse.c** : Lancement de wa.c sur la séquence à parser. Il gère également la lecture du fichier contenant toutes les séquences. Comme ce fichier peut atteindre plusieurs centaines de Mo il utilise un buffer pour sa lecture. Les fonctions de gestion du buffer se trouvent également dans ce fichier.
- **wascan-main.c** : Fichier qui permet de gérer l'interface ligne de commande, il gère les erreurs sur les options du programme ou sur les arguments passés en paramètres.
- **adn.c** : Gestion des nucléotides et acides aminés, ce fichier contient notamment les fonctions permettant de passer d'un motif nucléique à un motif protéique.

- **wa_io.c** : Gestion des entrées/sorties d'automates pondérés, ce fichier gère la lecture du fichier de l'automate pondéré.

Après une étude du code de Wapam, il s'est avéré que ce programme était une base convenable pour ce que l'on souhaitait réaliser. En effet on cherche à faire un parsing avec erreurs pour des protomates, et l'outil à notre disposition Wapam permet de faire un parsing mais pour une recherche de motif seulement. Il s'agit donc de le modifier afin de l'adapter pour qu'il ait de nouvelles fonctionnalités. Il faut également noter que Wapam est programmé en langage C ce qui m'était facile d'accès. De plus Wapam est un outil rapide et cela correspondait tout à fait au cahier des charges fixé.

3. Déroulement du Stage

La première partie de mon stage a consisté, comme je l'ai dit précédemment, à me documenter et à faire des choix pour savoir comment procéder à l'élaboration de ce programme. Une fois ces choix faits, j'ai commencé à programmer et à rédiger une documentation. Ensuite avec d'autres membres de l'équipe nous nous sommes penchés sur la mise en ligne du programme et la conception de l'interface Web. C'est donc de cela que je vais parler à présent.

Premièrement je détaillerai comment j'ai réalisé le parsing par un protomate ainsi que toutes les autres modifications apportées au programme Wapam, et deuxièmement nous verrons la mise en place de l'interface Web.

3.1. Modification du programme Wapam

J'ai donc fait le choix de réutiliser le programme Wapam. Ainsi j'ai cherché à compléter ce programme pour lui apporter de nouvelles fonctionnalités. La fonctionnalité principale qui lui est ajoutée est le parsing des séquences par un protomate : en effet Wapam cherche des motifs dans des séquences : il n'y a donc aucune notion d'état de début ou de fin, et on peut trouver plusieurs motifs par séquence. Avec la nouvelle fonctionnalité on cherche si les séquences vérifient ou non le protomate : il n'y a donc pas de possibilités de plusieurs résultats dans une même séquence. On conserve la possibilité de réaliser un parsing avec erreurs : on peut fixer un seuil pour élaguer les résultats. Un critère important dans le nouveau programme, on souhaite qu'il soit rapide, on fera donc attention à conserver la rapidité de Wapam.

Au début de nos modifications sur le code de Wapam, nous avons pris contact avec son concepteur Mathieu Giraud. On souhaitait lui demander si l'on pouvait intégrer mes nouvelles

fonctionnalités à son outil dans le but de créer un programme global regroupant les nouvelles et les anciennes options. Il a accepté et donc durant mon stage nous avons fait des réunions d'avancement régulières afin de lui faire part des modifications et d'avoir son aval.

3.1.1. Option protomate

Le but de ce stage était donc de réaliser le parsing avec erreurs d'une banque de données de séquences par un protomate. Au début de la conception du programme, l'option protomate et l'option recherche de motif étaient opposés au niveau des matrices de coût des automates. En effet le concepteur de Wapam avait prévu son code afin que les coûts soient maximisés. Les matrices de coûts étaient composés de nombres négatifs ou nuls. À l'inverse pour l'équipe s'occupant des protomates les matrices avaient des coûts positifs ou nuls et donc on devait minimiser le score. Dans un premier temps nous avons donc inversé les opérations pour que lorsqu'on sélectionne l'option protomate le score soit minimisé et pour qu'il soit maximisé avec l'option recherche de motif.

Une fois que le code était bien maîtrisé, les changements pour l'option protomate étaient assez simples. Il fallait empêcher que l'état initial soit actif ailleurs qu'au début de la séquence et vérifier que la fin de la séquence correspondait bien à l'état final de l'automate. Pour changer cela, j'ai modifié les fonctions d'initialisation afin que l'état initial ne puisse être actif qu'une seule fois au début de chaque séquence, et concernant la fin de la séquence j'ai écrit une fonction permettant de la détecter et de vérifier également que l'état final était actif. Après il suffisait de créer une option pour la ligne de commande. J'ai également changé les sorties afin de les adapter aux protomates. Cela m'a permis d'apprendre quelques notions de XML, une des sorties étant dans ce langage.

3.1.2. Tri des résultats

Après la mise en place de l'option permettant le parsing par un protomate, nous avons cherché à compléter au maximum l'outil créé. En effet, Wapam existait depuis quelques années et certains reproches lui étaient faits, on a donc essayé de palier les défauts de Wapam et de créer un outil plus complet. La première modification apportée à la structure même de Wapam a été la gestion des résultats.

À l'origine dès que l'on trouvait un résultat on l'affichait à l'écran. Il n'y avait donc pas de mémorisation. Or on s'est rendu compte que lors du traitement de banques de données importantes, il pourrait être intéressant de trier les résultats par score. En effet, on peut se retrouver avec un fichier résultat contenant des milliers de séquences, et sans tri, l'interprétation de ce fichier peut être délicate. Pour cela j'ai créé une structure « résultat » contenant par exemple le nom de la séquence ou le coût global, cela a permis une mémorisation des résultats dans un tableau de structures. Pour la fonction de

tri des structures, j'ai tout d'abord essayé de la programmer mais je me suis rendu compte que cela prenait beaucoup de temps à l'éxécution du programme car mon algorithme de classement n'était pas optimisé. Ainsi j'ai regardé ce qui existait dans les librairies et j'ai trouvé la fonction qsort de la librairie stdlib.h qui prend en paramètres, entre autres, un tableau de structures ainsi qu'une fonction de comparaison entre deux structures. Cette fonction permet un tri optimal et donc ne nous a pas handicapé au niveau du temps d'exécution. Cela nous a permis un affichage plus lisible mais le remaniement de la structure de données de résultats a été assez long à effectuer.

En effet, l'avantage de ne pas trier les résultats était que l'on n'avait pas besoin de faire d'allocation dynamique de mémoire ce qui allégeait le programme. En remaniant la gestion des résultats j'en ai eu besoin car on ne peut pas connaître le nombre de résultats à l'avance. De plus j'en avais également besoin pour stocker certains paramètres concernant des séquences résultats, comme des chaînes de caractères très longues. Toutes ces allocations dynamiques ont créé des bugs mémoires que j'ai mis un certain temps à corriger.

3.1.3. Assainissement du code

Certaines parties du code de Wapam n'étant pas très intuitives, j'ai donc essayé de les simplifier. Par exemple j'ai essayé de modulariser le code, en découpant des fonctions en sous fonctions. J'ai également rajouté de nombreux commentaires et j'ai créé une nouvelle documentation (cf annexe 1). Le remaniement du code m'a pris un certain temps car je testais au fur et à mesure toutes les fonctionnalités du programme afin de vérifier qu'elles n'étaient pas affectées par mes modifications. J'ai également appris à générer des documentations Doxygen.

Ensuite nous avons décidé d'intégrer plus en profondeur notre option : nous avons modifié le code afin que l'on puisse choisir de minimiser ou maximiser le score avec l'option protomate et l'option recherche de motif. Cela a permis de symétriser le code et de le rendre beaucoup plus propre. Par la suite nous nous sommes rendus compte que cela rajoutait beaucoup d'options dans la ligne de commande et que ce n'était pas la solution la plus simple. En effet au vu de la matrice de coûts incluse dans le fichier de l'automate (cf annexe 2), l'utilisateur du programme sait normalement déjà si il a besoin de minimiser ou de maximiser son score. Nous avons donc pensé qu'il était plus simple d'intégrer le type d'optimisation au fichier de l'automate vu que le lien entre l'optimisation et la matrice de coûts est direct.

3.1.4. Récupération du début et de la fin du motif

Un des reproches principaux que les biologistes faisaient à Wapam, était que lors d'une recherche de motif, le programme ne rendait pas d'indication sur l'emplacement du motif dans la séquence. Par conséquent, quand les séquences étaient longues et le motif petit, l'utilisateur ne pouvait

pas toujours se servir du score rendu par le programme de manière pertinente. C'est pour cela qu'on a voulu pouvoir récupérer le début et la fin du motif dans la séquence.

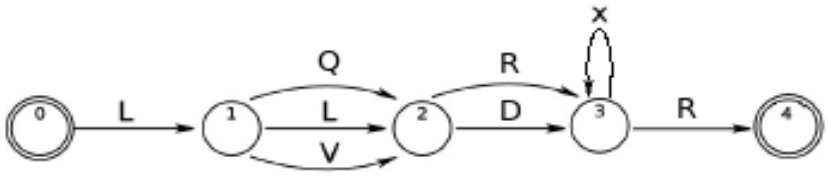
Pour trouver la fin du motif ce n'était pas difficile car lorsqu'on trouve un score on a toujours la possibilité de savoir où l'on est dans le fichier de séquences. Par contre vu qu'on ne garde pas le tableau des scores en entier mais qu'on ne garde que deux colonnes on ne peut pas remonter dans le tableau pour retrouver la position de début du motif. C'est pour cela que l'on a décidé de remplir notre tableau avec des couples d'entiers à la place des entiers. Cela permet de former une paire « début et score », avec le score qui évolue au fur et à mesure de la lecture de la séquence, mais qui garde toujours en mémoire sa position de début du motif. On a donc changé le code afin de créer à la place des tableaux d'entiers des tableaux de structures. Ensuite lors de la récupération d'une séquence résultat on pouvait avoir facilement accès au début du motif.

3.1.5. Option motif minimal-maximal (smallest-biggest pattern)

Nous avons également créé une option permettant de choisir si l'on souhaite avoir le motif le plus étendu ou le plus petit. En effet, si dans une même séquence on a deux résultats de même score et qui ont la même position de fin, alors on peut comparer la position de début du motif et rendre au choix le plus étendu ou le moins étendu. Cette option est pratique par exemple si on est en présence de gaps.

Exemple :

Soit l'automate suivant :



Et sa matrice de coûts correspondante :

Lettres: /Transitions:	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
0->1	3	3	3	3	3	3	3	3	3	3	0	3	3	3	3	3	3	3	3	3
1->2	3	3	3	3	3	0	3	3	3	3	0	3	3	3	3	3	3	3	3	0
2->3	3	0	3	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3->3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3->4	3	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Considérons les séquences suivantes :

>Séquence1

LVDAIDR

>Séquence2

LQKKKLQNCR

Résultats avec l'option motif maximal :

Séquence1 :	début du motif : 0	fin du motif : 6	coût : 0	**LVDAIDR**
Séquence2 :	début du motif : 0	fin du motif : 9	coût : 3	**LQKKKLQNCR**

Résultats avec l'option motif minimal :

Séquence1 :	début du motif : 0	fin du motif : 6	coût : 0	**LVDAIDR**
Séquence2 :	début du motif : 5	fin du motif : 9	coût : 3	LQKKK**LQNCR**

Dans la séquence 2, il y a 2 résultats possibles qui se terminent en position 9 : LQKKKLQNCR et LQNCR, si on lui donne l'option biggest match il trouvera le premier, et si on lui donne l'option smallest match il rendra le 2ème.

3.1.6. Option de mémorisation

La deuxième fonctionnalité importante que l'on a rajouté à Wapam, est l'option de mémorisation. Cette option permet de mémoriser les états par lesquels on est passé dans l'automate ainsi que de visualiser l'évolution du score (cela permet savoir à quel endroit de la séquence le score a changé).

Cette option utilise le même algorithme que nous avons vu précédemment. Cet outil a l'avantage de mettre en évidence les acides aminés qui sont incorrects et qui ont détérioré le score. Cela permet également de voir si le parsing est significatif (si on reste longtemps dans un gap par exemple).

Pour cette option on a donc besoin du tableau complet de taille t :
t = nombre_d_états*(longueur_de_la_séquence + 1).
A la fin du remplissage du tableau on connaît le score final ainsi que l'état final dans lequel on est placé, on peut ainsi remonter dans le tableau grâce à un algorithme de rétro propagation, reprenons

notre précédent exemple :

	0	1	2	3	4	5
État 0	0	0	0	0	0	
État 1		1	1	0	1	1
État 2			2	2	0	1
État 3			1	2	1	0

Les transitions sont caractérisées par un état de départ, un état d'arrivée et un tableau de 20 coûts (un coût pour chaque acide aminé). Au début de l'algorithme, on a donc un score, un état *e* et l'avancement dans la séquence *i*. On sait alors qu'on est dans la colonne *i*, à la ligne *e* ce qui nous donne notre positionnement exact dans le tableau. On boucle alors sur les transitions et on recherche celles qui ont pour état d'arrivée l'état *e*, on regarde à dans la colonne *i*-1 les scores aux lignes correspondant aux états de départ de ces transitions. Dans notre exemple on cherchait à minimiser le coût donc on va chercher parmi les transitions possibles celle qui nous donne le coût minimal. On mémorise son état de départ et on recommence une itération.

Cette option m'a pris beaucoup de temps à programmer car elle a demandé une très bonne maîtrise des allocations mémoire. De plus cette option m'a posé quelques problèmes car on devait conserver la possibilité de parser une banque de données de séquences nucléiques avec un automate en base protéique et cela était très délicat à gérer. En effet sur une base nucléique il y a 3 phases possibles, c'est à dire que l'on doit considérer pour une même séquence trois possibilités différentes : une séquence partant du premier acide nucléique, du deuxième acide nucléique ou du troisième. En effet puisqu'un acide aminé est formé de trois acides nucléiques, quand on a une séquence en base nucléique on ne sait pas où commence le premier acide aminé et donc on doit tester toutes les possibilités. De plus les séquences d'acides nucléiques ne sont pas toujours des multiples de 3 ce qui peut poser des problèmes à la fin de la lecture de la séquence.

3.1.7. Profiling et améliorations possibles

Pour voir si je pouvais améliorer la vitesse du programme j'ai effectué un profiling à l'aide de gprof. En effet on s'est rendu compte que lorsque l'automate est complexe et que la base de données est importante, le programme prend beaucoup de temps. On a donc vérifié avec gprof quelle partie du programme prenait le plus de temps. On est arrivé à la conclusion que lorsqu'on a un petit automate ou peu de séquences, ce sont les entrées-sorties qui prennent le plus de temps mais dès que les fichiers deviennent plus complexes alors le temps est principalement passé dans la boucle de calcul principale. La boucle est déjà très optimisée donc cela paraît difficile d'améliorer le temps d'exécution, cependant

il y a quelques pistes qu'il faudrait tester.

Une des solutions serait de pratiquer un élagage : si un chemin est en dessous/au dessus du seuil on ne l'exploite pas ce qui permettrait de réduire le nombre de branches de l'algorithme de Viterbi et permettrait une amélioration des performances. On peut également penser à faire une boucle sur le nombre d'états plutôt que sur le nombre de transitions dans le cas du parsing par protomate. En effet la boucle sur les états pourrait être plus intéressante que celle sur les transitions. Dans le cas de la recherche de motifs la boucle sur les états est généralement plus avantageuse car l'automate est assez linéaire. Au contraire en ce qui concerne le protomate ça pourrait être intéressant de faire une boucle sur les états. On le note comme une possible amélioration.

3.1.8. Résultats

Au final notre option de parsing à l'aide d'un protomate nous donne globalement les mêmes résultats en vitesse que Wapam. Pour donner un ordre d'idée si on passe en paramètre un fichier de séquences de 10 Mo et un automate de 350 états le programme s'exécutera en un peu moins de 3 minutes. De plus des tests ont été réalisés sur la banque de données européenne Swissprot et se sont bien déroulés.

En ce qui concerne l'option mémorisation, le problème est qu'elle ralentit le programme. En effet elle utilise beaucoup plus de ressources mémoire et dès que la base de données est importante elle devient difficile à utiliser. Quand les banques de données sont petites l'option de mémorisation met 2 fois plus de temps que l'exécution normale mais le ralentissement peut aller jusqu'à 6 fois le temps d'exécution normal. C'est pourquoi il faut prévenir l'utilisateur qu'avant de lancer une mémorisation, il peut être intéressant de réaliser dans un premier temps un scan normal, qui permettra de faire un tri dans les séquences qui l'intéressent et qui permettra de réduire le nombre de séquences passées en paramètre dans notre option.

3.2. Mise en place de l'interface Web

Une fois le programme fonctionnel, nous avons réfléchi à sa mise en ligne sur la plateforme de bio-informatique Genouest. Pour cela on devait définir le découpage en différentes pages, ainsi que des noms de programme intuitifs qui puissent guider l'utilisateur.

La première étape a été de faire un schéma qui résumait les différents liens entre Protomata Learner, Wapam et notre nouveau programme. En effet Wapam étant déjà en ligne sur la plateforme de bio-informatique, une des contraintes que l'on avait, était de modifier le moins possible son interface afin de ne pas déranger ses utilisateurs. Ensuite s'est posé le problème des noms des programmes : on souhaitait conserver le nom de Wapam pour que ses utilisateurs ne soient pas dépaysés. Seulement

Wapam (Weighted Automata PAttern Matching) implique l'idée de recherche de motif, et on pensait cela avait un côté réducteur. C'est donc pour cela que l'on a décidé de renommer notre programme Wascan pour Weighted Automata Scan. Toutes ces décisions ont été prises lors de réunions afin que chacun puisse donner ses arguments et que Mathieu Giraud puisse participer à l'évolution de son programme ainsi que nous donner son aval pour le changement de nom.

Une fois que le nom Wascan a été choisi, il restait encore le problème du découpage en plusieurs pages. Afin de clarifier l'utilisation de l'interface on a décidé de rendre complètement indépendant le module s'occupant de la recherche de motif de celui s'occupant du parsing par un protomate. Ainsi l'utilisateur de Wapam n'avait pas d'options supplémentaires sur sa page Web et cela évitait de surcharger l'interface. Le seul changement a donc été le nom de l'application : le nom présent sur l'interface est Wascan – Prosite (Prosite est un format d'automate pour la recherche de motif).

L'étape suivante a été de s'occuper de l'interface des programmes utilisant les protomates. On a donc décidé premièrement de réaliser une page donnant accès à l'outil Protoma Learner qui génère les protomates. De cette page cela nous paraissaît intéressant de mettre un lien direct vers Wascan avec l'option protomate afin de faciliter l'accès à cet outil qui est une suite logique à Protomata Learner. Le nom apparaissant sur l'interface est Protoscan, afin de bien spécifier que l'on travaille avec des protomates.

Et enfin il fallait s'occuper de créer une page pour l'option mémorisation car cette option n'est pas accessible directement. En effet on a décidé que l'utilisateur pourrait y accéder seulement après avoir eu la possibilité de faire un premier parsing, ceci dans l'optique de réaliser cette option sur un nombre de séquences assez réduit. Enfin nous nous sommes rendus compte que cela pourrait être intéressant de créer un programme prenant en paramètre les sorties de l'option mémorisation et les mettant en page dans une interface permettant de visualiser l'alignement des séquences en fonction de l'automate. Ce programme aurait pour nom Waalign, et son implémentation est donc un projet qui pourra être réalisé dans l'équipe postérieurement.

(cf page suivante pour le schéma de l'interface)

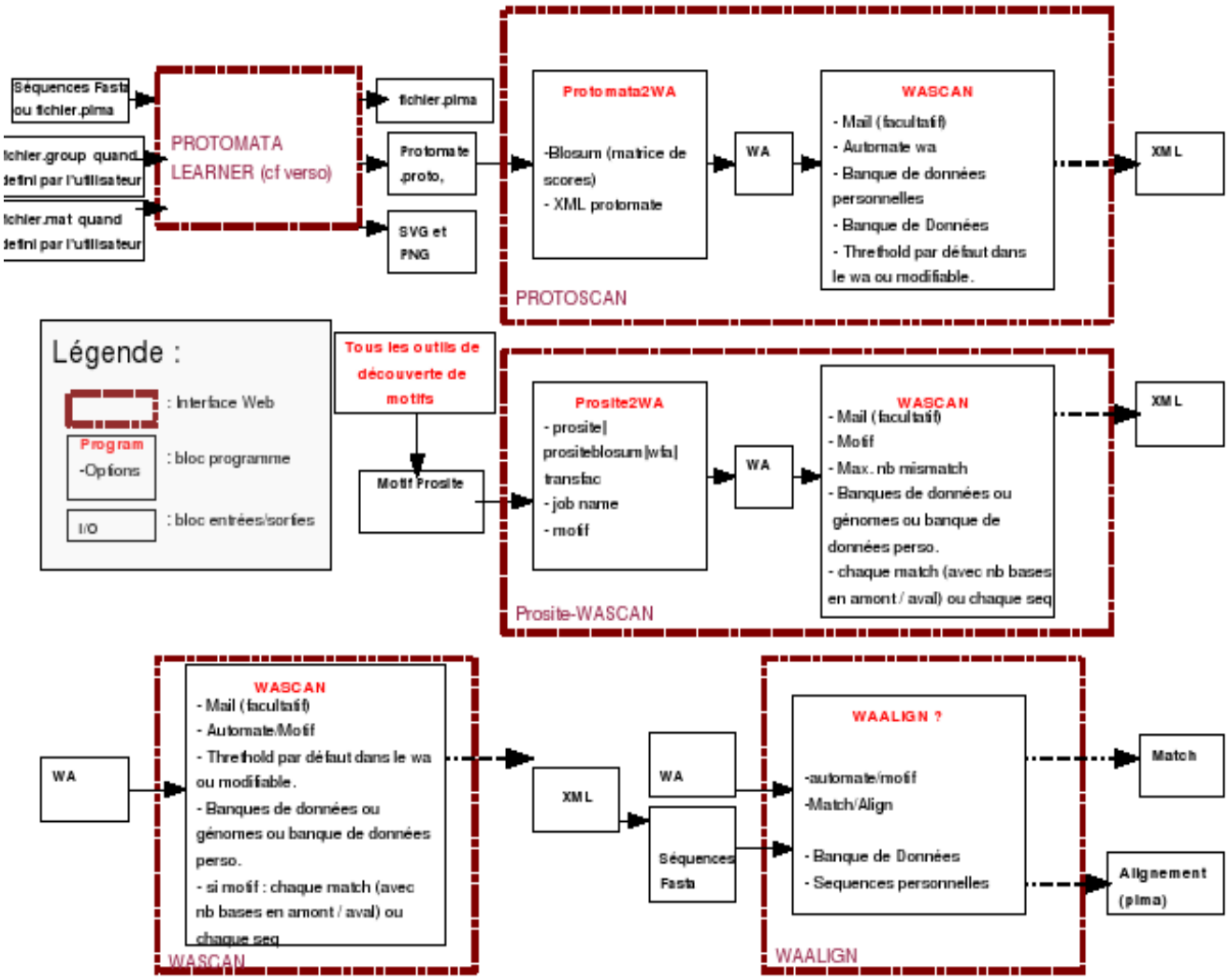


Illustration 1: Schéma du découpage en modules de l'interface Web

4. Conclusion

Le cahier des charges a été rempli et le programme est en train d'être mis en ligne, tout s'est donc déroulé comme prévu. De plus l'outil développé complète le programme Wapam et va offrir de nouvelles possibilités à ses utilisateurs. Cela n'était pas prévu initialement car l'outil devait être indépendant, finalement cela a permis la création d'outil plus complet et donc plus intéressant que ce soit pour les membres de l'équipe Symbiose ou pour les biologistes utilisateurs de Wapam.

Pour moi cela a été une première expérience professionnelle dans l'informatique et cela m'a permis de découvrir le fonctionnement d'un laboratoire de recherche public. J'ai également découvert le milieu de la bio-informatique, domaine d'application que j'ai trouvé très intéressant car très concret. De

plus, lors de ce stage, j'ai eu la chance de pouvoir gérer mon travail de façon autonome et cela a été très formateur que ce soit au niveau des contraintes de temps ou des responsabilités que cela implique. Ce stage a été aussi l'occasion de travailler en équipe, j'en retire que les capacités d'adaptation ainsi que la communication sont des qualités très importantes. De plus comme nous avions des réunions d'avancement régulières cela m'a permis d'apprendre à synthétiser, ainsi que à avoir un oeil critique sur mon travail. Au niveau théorique j'ai appris de nombreuses choses sur le développement dans un environnement Unix. En effet quand j'ai développé mon programme, j'ai travaillé sous Fedora Linux, j'ai donc appris un certain nombre de commandes en langage Shell afin de pouvoir compiler et debugger mon programme ainsi que pour pouvoir travailler sur le cluster de la plateforme de bio-informatique « Genouest ». J'ai également appris quelques bases concernant les makefile, ainsi qu'à utiliser un gestionnaire de version.

Cela m'a également permis de préciser mon projet personnel, tant en observant le fonctionnement de l'intérieur que par les échanges que j'ai pu avoir avec les membres de l'équipe. Finalement, ce stage restera pour moi un excellent souvenir, puisqu'il aura été l'occasion de nombreuses découvertes, tant au niveau technique qu'au niveau humain.

Références bibliographiques

- [1] http://genoweb.univ-rennes1.fr/Serveur-GPO/outils_acces.php3?id_syndic=250 (consulté le 01/10/2007)

- [2] Stéphane Guyetant, Mathieu Giraud, Ludovic L'Hours, Steven Derrien, Stéphane Rubini, Dominique Lavenier, and Frédéric Raimbault. Cluster of reconfigurable nodes for scanning large genomic banks. Parallel Computing, 2005

- [3] http://www.isima.fr/vbarra/IMG/pdf/HMM.pdf (consulté le 01/10/2007)

- [4] Mathieu Giraud Architectures reconfigurables pour la recherche par automates de motifs dans les séquences génomiques, 2005

Annexe 1 : Documentation Utilisateur de Wascan

WASCAN

Manuel utilisateur

Version <*0.0*>

HISTORIQUE :

Date	Version	Description	Partie(s) modifiée(s)
25/06/07		Programme de parsing de à l'aide d'un automate	

Table des matières

Introduction

Wascan est une évolution de Wapam, crée par Mathieu Giraud en 2004. Wapam est un outil de recherche de motifs mis en ligne sur le site de la plate-forme OUEST-genopole®. Wapam peut rechercher des motifs protéiques ou nucléiques, avec ou sans erreur(s), dans des génomes ou dans des banques de données. Wascan reprend toutes les fonctionnalités existantes de Wapam et lui en apporte de nouvelles. Les fonctionnalités principales ajoutées sont le parsing des séquences par un protomate et la mémorisation des états. En effet Wapam cherche des motifs dans des séquences : il n'y a donc aucune notion d'état de début ou de fin, et on peut trouver plusieurs motifs par séquence. Avec la nouvelle fonctionnalité de parsing par un protomate on cherche si les séquences vérifient ou non le protomate : il n'y a donc pas de possibilités de plusieurs matchs possibles. On conserve la possibilité de réaliser un scan avec erreurs : on fixe un seuil maximal et on rendra toutes les séquences qui ne correspondent pas au seuil. Wapam ne permettait pas non plus de suivre par quels états on passe dans l'automate, ni de voir l'évolution du coût.

Chaque séquence est enfilée progressivement dans un automate pondéré. Il en ressort un score qui permet d'évaluer l'adéquation de la séquence avec l'automate. Une recherche avec ou sans erreurs prend le même temps d'exécution.

Manuel d'utilisation

Récupération des sources

wa.c - Calcul principal pour le parsing.
-> L'algorithme utilisé dans ce fichier est l'algorithme de Viterbi.

parse.c - Lancement de wa.c sur la séquence à parser.

wascan-main.c - Interface ligne de commande.

adn.c – Gestion des nucléotides et acides aminés.

wa_io.c - Entrées/Sorties d'automates pondérés.

temps.c - Fonction de temps.

parse_aux.c -Fonctions auxiliaires du parsing

calcul_memo.c - similaire à wa.c mais pour l'option mémorisation

parse_memo.c - similaire à parse.c mais pour l'option mémorisation

Compilation

Makefile dans le répertoire « src ». On n'a pas besoin de recompiler le programme pour changer le seuil.

Exécution

Le mode d'exécution de Wascan est non-interactif.

Wascan considère un automate ou un protomate (wa_file, voir documentation de wa-tool)

et un fichier de séquences au format FASTA (seq_file).

./wascan [options] wa_file seq_file

-p Protomata version

-M Match

Parsing options:

-T Translation (proteic pattern / nucleic database)

-N No translation(proteic pattern / proteic database or nucleic pattern / nucleic database)

-r reverse strand

-m only the maximum hit per sequence (doesn't work with the protomata version)

-j only the smallest hit per sequence (doesn't work with the protomata version)

-i only the biggest hit per sequence (doesn't work with the protomata version)

-t # fix the threshold

Output formats (mandatory):

-x OUESTgenopole(R) XML

-z no sequence display in xml output

-f .bed (flat): one line per match

-s Synthetic results

Batch control:

-a #### log file

-k # kill after # results

-K #### kill file

Verbose output:

-v be verbose

-g display progress status

-b display progress bar (short)

-c display genetic code

-w display wa\n");

Help:

-h display this help

-V display version nunber

Interprétation des résultats

Un motif ou un protomate sera représenté par un automate, c'est à dire un ensemble d'états reliés entre eux par des transitions. L'automate est pondéré, c'est à dire que chaque transition est étiquetée par une lettre qui peut être lue selon l'alphabet de la séquence (bases nucléique ou protéique) et par un poids.

La séquence est progressivement « enfilée » dans l'automate, et, à chaque position, le poids de sa transition est additionné au score. Ce poids reflète l'adéquation d'une partie de la séquence cible

(banque ou génome) avec la lettre lue à cette position dans l'automate.

Le motif est reconnu lorsque l'état final est actif avec un score en adéquation avec le seuil

d'erreur fixé. Pour le protomate, une condition sera nécessaire en plus du score, on ne tiendra compte du score que si l'on est dans l'état final de l'automate et à la fin de la séquence.

Tout d'abord l'utilisateur doit faire les choix entre les options principales, c'est-à-dire : motif/protomate, match et/ou scan.

Attention, dans chaque cas (recherche de motifs et correspondance avec un protomate), un seuil est associé qui correspond à un certain seuil d'erreur. Cependant il ne signifie pas la même chose que ce soit pour l'option distance ou similarité :

-Option distance : le seuil t correspond au coût maximal acceptable entre le protomate et la séquence. On rendra donc toutes les séquences qui ont un résultat inférieur à t : on cherche à minimiser le score.

-Option similarité : le seuil t correspond au score minimal acceptable pour que l'on considère qu'il y a présence d'un motif : on cherche à maximiser le score.

Les options distance et similarité ne sont pas des options accessibles à l'exécution du programme, on connait l'option grâce à la lecture de l'automate.

On a également une **option biggest/smallest match** qui permet pour l'option motif de choisir si l'on souhaite avoir le motif le plus compact ou le plus grand possible (utile si présence de gaps).

Dans Wascan, il y a également une **option match**, qui permet de visualiser le chemin qui a été considéré par l'automate, cela permet de mettre en évidence des gaps par exemple. On peut également faire apparaître l'évolution du coût afin de trouver à quels endroits les substitutions ont été coûteuses.

Pour l'option match, les séquences données ne doivent pas dépasser 5 Mo.

Exemples d'utilisation

Fichier .wa :

Exemple d'un fichier WA :

WA or1

29 transitions

28 states

2 initial state(s) : 0 5

1 final state(s) : 27

default threshold 0

optimisation : min

	->	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(...)																					
25	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	27	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1

WA end

- Les lignes qui précèdent la matrice doivent être écrites exactement dans ce format,
- "or1" est le nom du WA.

Fichier fasta (.fas):

Exemple d'un fichier Fasta :

>Q9VTL1_CN12L_DROME

MFGTVNNYLSGVLHAAQDLDGESLATYLSLRDVHVQNHNLYIAQPEKLVDRFLKPPLDEVVSAHLKVLYHLA
QEPPGYMEAYTQQSAACGAVVRLLQQLKDENWCLPLMYRVCLDLRYLAQACEKHC

>Q58665_LIVH_METJA

MVKLLLKDLGEKMILEGAIIYSNLLVLLALGLTLTYITTNVPNFAQGSYAIVGSYVALTLLKLFGICPYLSLPVLF
VVGAIVGLITYLALKPLIKRNASVEILMIATLAIDLILLGVIGAYSEILSQ

>Q72MD8_LNT2_LEPIC

MDTLHHRFQQFQKTIWFNIFCYLWTGIFSFLAFAPVSLTHFVWIAPFGFFWLSLKYHGKYKKLFFHGLLIGVV
FYAISFHWIIHMAITFGNFPYVVAILILLFAGLLFGLKFPIFMMSFSFLSGKIGRHSVWVAGFCGLLSELIGPQLFP
WYWGNLAAGNIILAQNAEITGVYGISFLVFIVSYTLFQSNPWHWKEIIHSKEKRKQYLRFITLPALLLLTFIVSGI
FLFKKWENVKPVKSLNVLIVQPDAPLSF

>Q8EYY4_LNT2_LEPIN

MDTLHHRFQQFQKTIWFNIFCYLWTGIFSFLAFAPVSLTHFVWIAPFGFFWLSLKYHGKYKKLFFHGLLIGVV
FYAISFHWIIHMAITFGNFPYVVAILILLFAGLLFGLKFPIFMMSFSFLSGKIG

Après le « > » vient le nom de la séquence, et en dessous on a la séquence d'acides aminés (ou d'acides nucléiques).

Exemple avec option motif et sans mémorisation :

Automate : L -> V,Q,L -> D

chemin.wa :

->		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
0	1	3	3	3	3	3	3	3	3	3	3	0	3	3	3	3	3	3	3	3	3
1	2	3	3	3	3	3	0	3	3	3	3	0	3	3	3	3	3	3	3	0	3
2	3	3	3	3	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Séquences:

chemin.fas :

>GLPF_ECOLI_GLPp

LVDAIDR

>GLPF_PSEAE_GLPp

LQRDIDRLQRDIDRLVDAIDRLVDAIDR

>P04141_GMCSF_HUMAN

LQNKKKLQNCR

>P60568_IL2_HUMAN

LVQPIDR

Ligne de commande :

wascan -f chemin.wa chemin.fas

L'option -f est une option de sortie qui affiche un resultat par ligne.

Résultats :

GLPF_ECOLI_GLPp	0	2	#	or1	0
GLPF_PSEAE_GLPp	0	2	#	or1	0
GLPF_PSEAE_GLPp	7	9	#	or1	0
GLPF_PSEAE_GLPp	14	16	#	or1	0
GLPF_PSEAE_GLPp	21	23	#	or1	0

Comme on n'a pas précisé de seuil, on récupère le seuil par défaut de l'automate qui est ici de 0. Autre remarque, les 2 nombres qui suivent l'identifiant de la séquence sont le début et la fin du motif,

l'indexation se fait à partir de 0.

Exemple avec option motif et mémorisation :

Automate : L -> V,Q,L -> D

chemin.wa :

WA or1

3 transitions

4 states

1 initial state(s) : 0

1 final state(s) : 3

default threshold 0

optimisation : min

->		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V		
0	1	3	3	3	3	3	3	3	3	3	3	0	3	3	3	3	3	3	3	3	3	3	3
1	2	3	3	3	3	3	0	3	3	3	3	0	3	3	3	3	3	3	3	3	0	3	3
2	3	3	3	3	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Séquences:

chemin.fas :

>GLPF_ECOLI_GLPp

LVDAIDR

>GLPF_PSEAE_GLPp

LQRDIDRLVDAIDR

>P04141_GMCSF_HUMAN

LQNKKKLQNCR

>P60568_IL2_HUMAN

LVQPIDR

Ligne de commande :

wascan -fM chemin.wa chemin.fas

L'option -M est l'option qui permet d'afficher par quel chemin on est passé, ainsi que l'évolution du coût.

Résultats :

GLPF_ECOLI_GLPp 0 2 # or1 0

LVD

cout:

> 0 0 0

chemin:

> 0 1 2 3

GLPF_PSEAE_GLPp 0 2 # or1 0

LQR

cout:

> 0 0 0

chemin:

> 0 1 2 3

GLPF_PSEAE_GLPp 7 9 # or1 0

LVD

cout:

> 0 0 0

chemin:

> 0 1 2 3

Un résultat est structuré ainsi :

Identifiant de la séquence début du motif fin du motif # nom de l'automate coût final Motif

cout:

> 0 0 0 *//évolution du coût pour chaque transition*

chemin:

> 0 1 2 3 *//états suivis dans l'automate*

Ici on a toujours le seuil par défaut.

Exemple avec option motif et biggest /smallest match :

Automate : L -> V,Q,L -> D,R-> x (gap de taille indéterminée) ->R

chemin.wa :

->		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V		
0	1	3	3	3	3	3	3	3	3	3	3	0	3	3	3	3	3	3	3	3	3	3	
1	2	3	3	3	3	3	0	3	3	3	3	0	3	3	3	3	3	3	3	3	0	3	3
2	3	3	0	3	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	
3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	4	3	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	

WA end

Séquences:

chemin.fas :

>GLPF_ECOLI_GLPp

LVDAIDR

>P04141_GMCSF_HUMAN

LQKKKLQNCR

>P60568_IL2_HUMAN

LVQPIDR

Ligne de commande avec l'option biggest match :

wascan -if -t 100 chemin.wa chemin.fas

Ici on impose un seuil à 100, comme on est dans le cas où on cherche à minimiser le coût on rend tous les scores inférieurs à 100.

Résultats avec l'option biggest match :

GLPF_ECOLI_GLPp 0 5 # or1 0

P04141_GMCSF_HUMAN 0 9 # or1 3

P60568_IL2_HUMAN 0 5 # or1 3

Ligne de commande avec l'option smallest match :

wascan -jf -t 100 chemin.wa chemin.fas

Ici on impose un seuil à 100, comme on est dans le cas où on cherche à minimiser le coût on rend tous les scores inférieurs à 100.

Résultats avec l'option smallest match :

GLPF_ECOLI_GLPp 0 5 # or1 0

P04141_GMCSF_HUMAN 5 9 # or1 3

P60568_IL2_HUMAN 0 5 # or1 3

Dans la séquence P04141_GMCSF_HUMAN, il y a 2 matchs possibles qui se terminent en position 9 : LQKKKLQNCR et LQNCR, si on lui donne l'option biggest match -i il trouvera le premier, et si on lui donne l'option smallest match -j il rendra le 2ème.

Exemple avec option protomate et sans mémorisation :

Automate : L -> V,Q,L -> D->A

chemin.wa :

->		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
0	1	3	3	3	3	3	3	3	3	3	3	0	3	3	3	3	3	3	3	3	3
1	2	3	3	3	3	3	0	3	3	3	3	0	3	3	3	3	3	3	3	0	3
2	3	3	3	3	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	4	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Séquences:

chemin.fas :

>GLPF_ECOLI_GLPp

LVDA

>GLPF_PSEAE_GLPp

LQRDIDRLQRDIDRLVDAIDRLVDAIDR

>P04141_GMCSF_HUMAN

LVDAR

>P60568_IL2_HUMAN

LVQA

Ligne de commande :

wascan -pf -t 3 chemin.wa chemin.fas

L'option -p permet de faire le parsing avec un protomate

Résultats :

GLPF_ECOLI_GLPp 0 0 # or1 0

P60568_IL2_HUMAN 0 0 # or1 3

L'important est que l'on soit à la fin de la séquence quand on est à l'état final.

Exemple avec option protomate et avec mémorisation :

Automate : L -> V,Q,L -> D->A

chemin.wa :

->		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
0	1	3	3	3	3	3	3	3	3	3	3	0	3	3	3	3	3	3	3	3	3
1	2	3	3	3	3	3	0	3	3	3	3	0	3	3	3	3	3	3	3	0	3
2	3	3	3	3	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3	4	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Séquences:

chemin.fas :

>GLPF_ECOLI_GLPp

LVDA

>GLPF_PSEAE_GLPp

LQRDIDRLQRDIDRLVDAIDRLVDAIDR

>P04141_GMCSF_HUMAN

LVDAR

>P60568_IL2_HUMAN

LVQA

Ligne de commande :

wascan -pfM -t 3 chemin.wa chemin.fas

Résultats :

GLPF_ECOLI_GLPp 0 4 # or1 0

LVDA

cout:

> 0 0 0 0

chemin:

> 0 1 2 3 4

P60568_IL2_HUMAN 0 4 # or1 3

LVQA

cout:

> 0 0 3 3

chemin:

> 0 1 2 3 4

Un résultat est structuré ainsi :

Identifiant de la séquence début de la séquence fin de la séquence # nom du WA coût final Séquence

cout:

> 0 0 0 *//évolution du coût pour chaque transition*

chemin:

> 0 1 2 3 *//états suivis dans l'automate*

Conclusion

Améliorations possibles et remarques :

- Dans le cas du parsing par protomate, l'élagage serait possible si dans la boucle critique on bouclait sur les états et non les transitions.

- Rétablir l'affichage des séquences quand on est en dehors de l'option match (l'affichage actuel des séquences ne permet pas l'affichage des séquences de plus de 5 Mo = taille du buffer)

- option biggest/smallest match n'existe pas avec l'option mémorisation

Annexe 2 : Formats des fichiers à traiter

Le programme à réaliser prendra en paramètres deux fichiers : un fichier d'extension WA (Weighted Automata) qui correspond à l'automate et un fichier de format « fasta » qui contient les séquences à analyser.

Structure d'un fichier WA :

Au début du fichier on trouve le nom du WA. Après vient la description de l'automate : nombre de transitions, nombre d'états, nombre d'états initiaux, nombre d'états finaux, seuil par défaut (en effet le parsing accepte les erreurs et le seuil permet de fixer combien d'erreurs on souhaite tolérer) et le type d'optimisation (c'est à dire si l'on cherche à minimiser ou maximiser le score). La dernière partie du fichier WA est la matrice de coûts qui permet de connaître tous les coûts des substitutions pour chaque transition.

Structure d'un fichier Fasta :

Exemple :

>Q9VTL1_CN12L_DROME

MFGTVNNYLSGVLHAAQDLDGESLATYLSLRDVHVQNHNLYIAQPEKLVDRFLKPPLDEVVSAHLKVLYHLA
QEPPGYMEAYTQQSAACGAVVRLLQQLKDENWCLPLMYRVCLDLRYLAQACEKHC

>Q58665_LIVH_METJA

MVKLLLKDLGEKMILEGAIIYSNLLVLLALGLTLTYITTNVPNFAQGSYAIVGSYVALTLLKLFGICPYLSLPVLF
VVGAIVGLITYLALKPLIKRNASVEILMIATLAIDLILLGVIGAYSEILSQ

>Q72MD8_LNT2_LEPIC

MDTLHHRFQQFQKTIWFNIFCYLWTGIFSFLAFAPVSLTHFVWIAPFGFFWLSLKYHGKYKKLFFHGLLIGVV
FYAISFHWIIHMAITFGNFPYVVAILILLFAGLLFGLKFPIFMMSFSFLSGKIGRHSVWVAGFCGLLSELIGPQLFP
WYWGNLAAGNIILAQNAEITGVYGISFLVFIVSYTLFQSNPWHWKEIIHSKEKRKQYLRFITLPALLLLTFIVSGI
FLFKKWENVKPVKSLNVLIVQPDAPLSF

Après le « > » vient le nom de la séquence, et en dessous on a la séquence d'acides aminés (ou d'acides nucléiques).