Algorithmes de reconstruction de séquences de peptides (rapport de TER)

Juanfer Mercier*, Guillaume Fertin[1,**], Géraldine Jean[2,**], Émile Benoist[3,**]

Nantes Université – UFR Sciences et Techniques
2 rue de la Houssinière, 44322 Nantes (FRANCE)

Abstract

Dans le contexte de travaux réalisés dans l'équipe ComBi du LS2N (Laboratoire des Sciences du Numérique de Nantes) et dans l'équipe BIA de l'INRAE de Nantes sur la spectrométrie de masse, nous cherchons à résoudre un problème de reconstruction de séquences peptidiques à partir d'informations partielles et parfois erronées, fournies en entrée sous la forme de séquences que nous appelons "*baitModels*". Les *baitModels* sont des séquences composées de trois types d'éléments : (1) des caractères représentant des acides aminés, (2) des valeurs numériques entre crochets représentant des masses et (3) des caractères entre crochets. Les crochets indiquent qu'il y a eu une modification (insertion, suppression ou substitution) d'un ou plusieurs acide(s) aminé(s). Les *baitModels* peuvent représenter une même séquence d'acides aminés (i.e. un peptide), potentiellement à quelques erreurs près. Il conviendrait, sur la base des *baitModels* fournis, de reconstruire le peptide en utilisant ou en ignorant les informations qu'ils portent. Dans ce rapport, deux méthodes de fusion des *baitModels* sont présentées : la première utilise des algorithmes d'alignement de séquences multiples et la seconde utilise des curseurs qui sont avancés itérativement. Étant donné que les algorithmes d'alignement utilisés sont issus de la littérature et que nos méthodes sont fondées sur des travaux antérieurs, dans un premier temps, un état de l'art succinct est présenté. Ensuite, les deux méthodes que nous avons implémentées sont détaillées. Enfin, ce rapport rapporte les résultats numériques obtenus avec ces méthodes sur un jeu de données issu du protéome humain.

Keywords: Séquences, Algorithmes, Fusion, Acides aminés, Peptides

∗. Auteur
∗∗. Encadrant
Email addresses: `juanfer.mercier@etu.univ-nantes.fr` (Juanfer Mercier), `guillaume.fertin@univ-nantes.fr` (Guillaume Fertin), `geraldine.jean@univ-nantes.fr` (Géraldine Jean), `emile.benoist@univ-nantes.fr` (Émile Benoist)
1. Professeur des universités, LS2N/Nantes Université
2. Maître de conférences, LS2N/Nantes Université
3. Doctorant, LS2N/Nantes Université

1. Introduction

1.1. Les protéines dans le vivant

Un organisme est un système vivant complexe composé d'une grande variété de molécules ayant des fonctions spécifiques. L'ADN (acide désoxyribonucléique) et l'ARN (acide ribonucléique) sont les supports de l'information génétique. Les protéines sont des enchaînements d'acides aminés (nommés résidus lorsqu'ils sont impliqués dans une séquence de protéine) produits selon l'information génétique, et sont les acteurs directs de fonctions très variés au sein d'un organisme. L'ensemble des protéines exprimées dans un échantillon biologique (e.g. cellule, organisme) à un instant donné constitue son protéome [1, 2].

Grâce à un processus connu sous le nom d'épissage alternatif, un gène donné peut être à l'origine de la production de plusieurs protéines. Il est donc difficile de savoir quelles protéines sont présentes dans un échantillon grâce à la seule information génétique et, par conséquent, le protéome doit être étudié directement pour obtenir des informations précises sur le fonctionnement de l'échantillon. De plus, les protéines peuvent porter des modifications chimiques ce qui complexifie l'étude du protéome par rapport à l'étude du génome (l'ensemble de l'information génétique contenue dans l'ADN) [3]. En effet, les protéines peuvent subir, lors de leur maturation, l'ajout d'une PTM ("*Post-Translational Modification*"), c'est-à-dire que des groupements chimiques peuvent être ajoutés ou supprimés de leurs résidus ; cette modification a un impact sur la fonction de la protéine, qui dépend de la modification et de son emplacement sur la protéine.

Lors de l'étude des protéines, il est donc nécessaire d'identifier les PTM (i.e. la nature des modifications) et de les localiser (i.e. l'emplacement des modifications) pour connaître précisément la fonction d'une protéine donnée. Ce problème est essentiel en médecine et dans des domaines comme l'alimentation. L'étude du protéome, la protéomique, permet une meilleure compréhension du fonctionnement cellulaire à partir de l'expression protéique et une technique permet d'explorer le protéome efficacement : la spectrométrie de masse.

1.2. La spectrométrie de masse

La spectrométrie de masse ("*mass spectrometry*" ou MS) est une méthode d'analyse qui consiste à détecter et identifier des molécules chargées (ions) à partir de leurs masses. À la fin des années 80, suite à des progrès dans le domaine de l'ionisation des molécules de taille importante, la MS a pu être appliquée à la protéomique [4]. La MS permet d'identifier des protéines et de séquencer des peptides (fragments de protéine) rapidement. En raison du grand nombre de données que la MS permet d'obtenir, des méthodes informatiques sont utilisées en protéomique. Lors d'une analyse par spectrométrie de masse, les protéines sont séparées en peptides et les peptides obtenus sont eux-mêmes fragmentés dans l'appareil. Le résultat obtenu est appelé *spectre de masse*. Un spectre de bonne qualité contient l'information de séquence en résidus du peptide qui l'a généré. Pour obtenir cette information de séquence, les spectres expérimentaux (produits par l'appareil) peuvent être comparés, à l'aide d'un score de similarité, à une base de données de spectres de masse théoriques

construits à partir d'une base de données de peptides. Étant donné que les mécanismes de fragmentation sont connus, il est possible de créer un spectre théorique "idéal" à partir d'un peptide donné. Cette étape de comparaison permet de produire un ensemble de PSM ("*Peptide-Spectrum Matches*") où chaque spectre expérimental est associé à un spectre théorique, et donc à un peptide ; le peptide qui a produit le spectre peut ensuite être identifié à l'aide de la séquence du peptide candidat sélectionné. Les peptides identifiés grâce à ces spectres permettent d'inférer quelles protéines sont présentes dans l'échantillon donné.

Néanmoins, lorsque les peptides portent des PTM, l'identification de leurs spectres est plus difficile. Il faut non seulement sélectionner le bon peptide candidat, mais aussi être capable d'interpréter le PSM afin de déterminer la nature et l'emplacement des modifications éventuelles qui séparent le peptide candidat de celui qui a généré le spectre ; il est ensuite possible de reconstruire la séquence du peptide, et donc de dire que le spectre est identifié. Nos travaux prolongent une méthode présentée dans la thèse d'Albane Lysiak [5]. Cette thèse propose des méthodes informatiques visant à identifier des spectres de masse issus de protéines modifiées et, plus particulièrement, lorsque ces protéines portent des PTM à la fois multiples et non connues *a priori*.

1.3. Travaux préalables et structure du rapport

Dans la thèse susmentionnée, un algorithme nommé `SpecGlob` est présenté. Essentiellement, cet algorithme tente d'aligner un spectre de masse théorique (appelé "*hit*") sur un spectre de masse expérimental (appelé "*bait*"). En comparant les deux spectres, il vérifie si les deux spectres sont alignés et s'il existe des décalages de masse (exprimée en Dalton, symbole Da) entre des résidus de protéines dû à des PTM. L'algorithme fournie une chaîne de caractères (appelée "*hitModified*") indiquant pour chaque résidu du *hit* s'il est aligné dans le *bait*, et si oui, si un décalage de masse doit être inséré pour aligner le résidu sur le *bait* (e.g. GGSQTI[Y]R[407,26]). Ensuite, il est possible d'interpréter le spectre expérimental à partir de l'alignement *hitModified* et les décalages de masse introduits dans l'alignement sont interprétés en séquences de résidus quand cela est possible. Cette interprétation (donnée sous la forme d'une séquence) est ce que nous appelons *baitModel* (e.g. GGSQTI[570.32]R).

Selon la nature et l'emplacement des PTM, il est possible d'obtenir plusieurs *baitModel* différents censés représenter un même peptide. Par conséquent, nous cherchons à reconstruire la séquence du peptide original en fusionnant les *baitModels*. Pour cela, nous cherchons à tirer parti des informations portées par les *baitModels* tout en ignorant potentiellement certaines sections que nous considérerons alors comme fausses. Dans ce rapport, nous présentons deux méthodes de fusion des *baitModels*. La section 2 est dédiée à un bref état de l'art des méthodes utilisées et implémentées dans la thèse ainsi que des méthodes d'alignement de séquences multiples utilisées par l'une de nos méthodes de fusion. Ensuite, nous présentons nos méthodes dans la section 3 avant de discuter des résultats obtenus avec celles-ci, section 4, sur un jeu de données issu du protéome humain. Enfin, ce rapport se termine par une conclusion générale des travaux que nous avons réalisés lors de ce travail d'étude et de recherche (TER).

2. État de l'art

Lorsque les peptides portent des PTM, l'identification de leurs spectres est plus difficile. Les méthodes dites OMS ("Open Mass Search") ont été développées pour traiter ce problème [5]. Elles sont capables d'identifier un grand nombre de spectres modifiés, mais elles se basent souvent pour cela sur des PTM connues à cause du temps de calcul requis pour considérer un grand nombre de PTM. Les méthodes OMS sont des outils de comparaison à une base de données de spectres théoriques. En d'autres mots, chaque spectre expérimental obtenu par la MS est comparé aux spectres théoriques générés à partir des peptides d'une base de données. Lors de la comparaison, un score de similarité est utilisé pour mesurer la ressemblance entre le spectre expérimental et le spectre théorique. Après cette comparaison, à chaque spectre expérimental sera assigné un ou plusieurs spectre(s) théorique(s) (i.e. un ou plusieurs peptide(s)) qui lui ressemblent d'après le score, et chaque couple spectre-peptide est nommé "Peptide-Spectrum Match" (ou PSM, voir FIGURE B.1). Les PSM renvoyés par un outil d'identification permettent d'obtenir, grâce aux peptides candidats identifiés, des informations sur l'identité des peptides qui correspondent aux spectres analysés.

La méthode OMS `SpecOMS`, développée dans le cadre d'une thèse LS2N/INRAE [6], permet d'identifier des spectres (i.e. des peptides) rapidement et a été utilisée par Lysiak lors de sa thèse pour obtenir des PSM à partir du protéome humain [5]. Cependant, selon une étude présentée au Chapitre 3 de la thèse de Lysiak, `SpecOMS` interprète difficilement les PSM lorsqu'ils comportent plusieurs modifications séparant le spectre théorique du spectre expérimental. Dans le contexte de la thèse et pour être capable de mettre en évidence des modifications multiples et sans *a priori* sur leurs nature, l'algorithme `SpecGlob` a été développé.

Dans cette section, nous présentons le logiciel `SpecOMS` puis le logiciel `SpecGlob`. Enfin, nous présentons les algorithmes d'alignement de séquences multiples ("Multiple Sequence Alignment" ou MSA) utilisés par l'une des méthodes que nous présentons dans la section 3.

2.1. Le logiciel `SpecOMS`

`SpecOMS` est un logiciel d'identification de spectres en mode OMS, développé dans le cadre d'une thèse dans les équipes ComBi (LS2N) et BIA (INRAE). Il est capable de comparer tous les spectres d'une base les uns aux autres grâce à un score appelé SPC ("*Shared Peaks Count*"), le nombre de pics avec des masses identiques entre deux spectres. Nous appelons *masse parente* la masse du peptide fragmenté, associée au spectre correspondant. `SpecOMS` est capable de calculer le SPC entre chaque couple de spectres au sein d'un ensemble de spectres donné, sans limite de différence de masse parente entre les spectres lorsqu'ils sont comparés. Le logiciel prend en entrée un ensemble de spectres expérimentaux à identifier ainsi qu'une base de données de protéines dont les peptides permettront de générer les spectres théoriques. De plus, il faut aussi tenir en compte des protéines qui peuvent contaminer un échantillon dans le laboratoire (e.g. kératine humaine, protéines de gants de latex) donc `SpecOMS` prend en entrée une base de protéines contaminantes. Enfin, l'utilisateur doit préciser un paramètre t correspondant au seuil de SPC à partir duquel il souhaite obtenir

un PSM. Chaque PSM contient les informations suivantes : identifiant du spectre (expérimental), séquence du peptide (ou spectre théorique), SPC, et Δm, qui est la différence de masse parente entre les deux spectres du PSM. Nous utilisons dans la suite de ce rapport le vocabulaire utilisé dans la thèse de Lysiak. Nous appelons "*hit*" le peptide (et par extension, le spectre) qui joue le rôle de spectre théorique dans le travail d'identification effectué par `SpecOMS`. Nous appelons "*bait*" le peptide (et par extension, le spectre) qui joue le rôle de spectre expérimental dans le travail d'identification effectué par `SpecOMS`.

Le logiciel est composé de plusieurs modules (voir FIGURE B.2) :

- À partir des spectres en entrée, la structure ***SpecTrees*** [6] est construite.
- Le module ***SpecXtract*** parcourt ensuite la structure ***SpecTrees*** ce qui permet d'extraire tous les PSM avec un SPC au moins égal au seuil t. Après cette étape, chaque spectre expérimental en entrée a 0, 1 ou n spectre(s) théorique(s) (issus des peptides candidats) qui lui sont associés, car ces spectres partagent avec lui au moins t pics.
- Enfin, optionnellement, le module ***SpecFit*** peut être utilisé pour sélectionner au plus un seul spectre théorique pour chaque spectre expérimental en utilisant la valeur du SPC ou un algorithme appelé ***shift*** (voir ci-après).

Si l'utilisateur le souhaite, `SpecOMS` est capable d'utiliser Δm pour générer un nouveau score de comparaison, une variante du SPC nommée *shift SPC* et les PSM seront produits sur la base de ce score. Pour obtenir ce score, l'algorithme ***shift*** peut être utilisé. Cet algorithme tente de réaligner deux spectres (i.e. déplacer certaines masses d'un spectre pour qu'elles soient égales à celles de l'autre, révélant ainsi une similarité) en partant de l'hypothèse qu'ils sont séparés par une seule modification de masse Δm.

L'étude, menée par Lysiak, des PSM produits par `SpecOMS` à partir des spectres théoriques [5] montre qu'il serait intéressant de séparer Δm en plusieurs parties afin de considérer plusieurs modifications de masses différentes et dont la somme correspond à Δm. Pour être capable de mettre en évidence des modifications à la fois sans *a priori* et multiples, `SpecGlob` a été développé.

2.2. Le logiciel ***SpecGlob***

`SpecGlob` est un algorithme qui repose sur l'alignement d'un peptide sur le spectre à identifier, le peptide et le spectre appartenant à un PSM (fournie par n'importe quelle méthode OMS) pour mettre en évidence les modifications qui les séparent ; si ces modifications sont correctement identifiées dans le peptide, la séquence du peptide qui a générer le spectre peut être retrouvée. Ainsi, `SpecGlob` va tenter de retrouver la séquence du peptide qui a produit le spectre en transformant le peptide candidat (dont la séquence est connue) afin de retrouver la séquence peptidique du spectre expérimental. En sortie de l'algorithme, nous nous attendons donc à obtenir la séquence du peptide enrichie des modifications placées correctement sur ses résidus (i.e. acides aminés). Dans cette séquence, la somme des masses des modifications devra correspondre à Δm, séparée en plusieurs masses si plusieurs modifications séparent le peptide identifié et le peptide qui a produit le spectre à identifier.

Étant donné que le PSM est le résultat d'une recherche OMS, les peptides du *bait* et du *hit* partagent probablement une certaine similarité en termes de résidus en commun puisque leurs spectres respectifs ont un certain nombre de pics avec des masses identiques. `SpecGlob` va alors essayer d'aligner les résidus du *hit* (représentés par une différence de masse entre deux masses successives du spectre) sur ceux du *bait* (représentés par une différence au sein d'une paire de masses donnée dans le spectre) en autorisant des décalages de masses. Ainsi, l'algorithme est censé trouver quels décalages de masse éventuels doivent être fait pour obtenir un alignement optimal des résidus du *hit* sur ceux du *bait* (i.e. les décalages de masse correspondront aux modifications éventuelles à réaliser dans le *hit* pour obtenir la séquence du *bait*).

À la fin de l'alignement effectué par `SpecGlob`, le Δm est découpé si nécessaire en plusieurs décalages de masses dont le nombre et les valeurs ne sont pas prédéfinis. Cet alignement est alors considéré comme optimal et il est donné sous la forme d'une chaîne de caractères appelée *hitModified*. Cette chaîne de caractères se termine souvent par un résidu K ou R car l'enzyme utilisée par le spectromètre pour fragmenter les peptides, la trypsine, coupe les protéines après chaque acide aminé K ou R. Le *hitModified* correspond à la séquence du *hit* enrichie d'indications spécifiques marquant les modifications à effectuer dans le *hit* pour obtenir le *bait*. Pour chaque PSM, le *hitModified* spécifie l'alignement de chaque résidu du hit. Il existe trois possibilités [5] :

1. Deux masses consécutives du *hit* (qui correspondent à la masse d'un résidu) sont alignés avec deux masses du *bait* sans avoir besoin d'insérer un décalage de masse ; dans ce cas, ce résidu est considéré comme retrouvé dans le *bait* et est indiqué tel quel dans le *hitModified*.
2. La différence entre deux masses consécutives du *hit* est trouvée entre deux masses du *bait*, mais l'alignement de ces masses requiert l'insertion d'un décalage de masse ; dans ce cas, le résidu du *hit* est écrit dans le *hitModified*, précédé de la valeur (en Da) du décalage de masse entre crochets.
3. Si la différence de masse entre deux masses consécutives du *hit* n'est pas utilisée dans l'alignement, cela signifie que le résidu courant est considéré comme absent et celui-ci est alors écrit dans le *hitModified* entre crochets.

Le *hitModified* indique des décalages de masse et des résidus non retrouvés qui forment les opérations (délétions, insertions, substitutions) à effectuer. Certaines de ces modifications permettent de retrouver la séquence du *bait*. G[I]T[-14.02]ACCITK est un exemple de *hitModified* fournit par `SpecGlob` à partir d'un PSM. Voici l'interprétation de cette séquence :

- T doit être décalé sur I, qui n'est pas retrouvé dans le *bait*, et ce décalage doit être de 14.02 Da.
- Or la masse de I (113.08 Da) moins 14.02 Da vaut 99.06 Da (la masse de V).
- I peut donc être substitué par V et nous obtenons la séquence GVTACCITK.

Ainsi, certaines modifications du *hitModified* peuvent être interprétées, résultant en une chaîne de caractères simplifiée. Plus précisément, des modifications non ambiguës peuvent être transformées en séquences car les masses des résidus (i.e. les masses des acides aminés) sont connues. Soit m la somme des masses des résidus (entre crochets) d'une modification et d le décalage éventuel qui suit les résidus de la modification (d peut être nul si aucun décalage ne suit les résidus de la modification). Les modifications peuvent être classées en trois catégories selon leur ambiguïté [5] :

- Une modification est classée comme **Verte** si elle peut être transformée en une séquence sans ambiguïté. Une délétion est caractérisée par un ou plusieurs résidus non retrouvés (entre crochets) dont la somme des masses est m et où $d = -m$. Une délétion peut être réalisée sans ambiguïté et est donc toujours classé comme une modification Verte. Pour une substitution, si m et d sont non nuls alors la modification est Verte si $m + d$ correspond à la masse d'un seul résidu et que cette masse ne correspond pas à la masse d'une séquence d'au moins deux résidus. Enfin, pour une insertion (c'est-à-dire pour une modification où $m = 0$ et $d \neq 0$), la modification est Verte si d correspond à la masse d'un seul résidu et que cette masse ne correspond pas à la masse d'une séquence d'au moins deux résidus.
- Une modification (substitution ou insertion) est classée comme **Orange** si $m + d$ correspond à la masse d'une séquence de 2 à n résidus (n étant un nombre arbitraire que nous fixons à 8 dans notre cas). Une modification Orange correspond à l'insertion d'une combinaison d'au plus n résidus mais pour laquelle l'ordre des résidus à insérer est incertain. De plus, plusieurs combinaisons de résidus peuvent avoir la même masse ($m + d$) ce qui justifie l'ambiguïté de ces modifications. Bien que les modifications de cette catégorie peuvent correspondre à plusieurs ensembles distincts de résidus, elle reste tout de même intéressante. En effet, nous pouvons par exemple pré-calculer une table contenant les masses de toutes les séquences d'au plus n résidus et l'utiliser pour déterminer quelles combinaisons de résidus sont susceptibles d'être présentes dans le *bait*.
- Toute autre modification est classée comme **Rouge** (e.g. $m + d$ correspond à la masse d'une séquence de plus de n résidus).

La séquence obtenue en simplifiant les opérations non ambiguës indiquées dans le *hitModified* est censée représenter le plus précisément possible la séquence du *bait*. Cette séquence simplifiée est l'interprétation du *bait* à partir de l'alignement *hitModified* et nous l'appelons *baitModel*. De plus, chaque *bait* peut avoir plus d'un *hit* qui lui sont associés. Par conséquent, chaque *bait* peut avoir plusieurs *hitModifieds* (et par conséquent plusieurs *baitModels*) qui lui sont associés et qui sont censés le représenter. Notre objectif est de reconstruire la séquence du *bait* en fusionnant les *baitModels*.

Nous présentons dans la section 3 deux méthodes de fusion des *baitModels*. Étant donné que nos méthodes sont censées prendre en entrée au moins deux *baitModels* et que ces derniers sont des séquences, une première idée est d'utiliser des algorithmes d'alignement de séquences multiples pour aligner les *baitModels* pour ensuite déterminer un consensus entre les différentes séquences. Nous présentons ci-après deux de ces algorithmes.

2.3. Algorithmes d'alignement de séquences multiples

En bio-informatique, l'alignement de séquences est une manière de représenter deux ou plusieurs séquences biologiques (e.g. ADN, protéines) les unes sous les autres, de manière à en faire ressortir les régions homologues ou similaires. En effet, si deux séquences sont alignées (i.e. similaires) il se peut qu'elles remplissent des fonctions similaires. Il existe dans la littérature plusieurs approches pour résoudre le problème d'alignement selon le type d'alignement voulu. Nous présentons dans cette sous-section quelques types d'alignement, un algorithme d'alignement pair-à-pair (i.e entre deux séquences) ainsi que deux méthodes heuristiques pour l'alignement de séquences multiples ("Multiple Sequence Alignment" ou MSA).

2.3.1. Quelques types d'alignement

Les caractéristiques d'un alignement dépendent fortement du type d'alignement utilisé. Nous présentons ici trois types d'alignement :

1. L'**alignement global** consiste à aligner l'intégralité de deux (ou plus) séquences. En d'autres mots, les séquences sont alignées de sorte à maximiser le nombre de lettres alignées. Cet alignement est adapté aux séquences étroitement liées [7]. Il est plus stable lorsque les sections de forte similarité sont déjà plus ou moins bien alignées.
2. L'**alignement local** consiste à aligner des sections ayant une forte similarité. Cette méthode aligne des sections de séquences et est adapté aux séquences ayant peu de similarités entre elles [7]. Cet alignement est plus stable que l'alignement global lorsque les sections de forte similarité sont peu ou pas alignées.
3. Les différences entre l'alignement global et l'alignement local ainsi que leur avantage respectif ouvre la possibilité de créer une méthode combinée permettant de générer des alignements plus précis [7]. L'**alignement semi-global** (ou "glocal") est une méthode hybride entre l'alignement global et l'alignement local alignant les séquences de sorte à obtenir le meilleur alignement partiel des séquences [8, 9]. Cet alignement ignore les écarts au début et/ou à la fin d'un alignement. Il est notamment utile si une séquence est beaucoup plus courte qu'une autre (e.g. lors d'une comparaison entre un peptide et un protéome). Dans ce cas, la séquence la plus courte devrait être alignée globalement (dans son entièreté) mais la séquence la plus longue devrait être alignée localement (partiellement).

Les méthodes d'alignement pair-à-pair sont utilisées pour évaluer la similarité entre deux séquences. Bien que limitées à deux séquences, les algorithmes d'alignement pair-à-pair sont utilisés par certaines méthodes d'alignement multiple et font partie des premiers algorithmes développés pour le problème d'alignement. L'algorithme d'alignement global Needleman-Wunsch [10, 9] et l'algorithme d'alignement local Smith-Waterman [11] sont deux méthodes d'alignement pair-à-pair. Ces méthodes sont implémentées avec des notions de programmation dynamique. L'algorithme Needleman-Wunsch est utilisé par `ClustalW` (une méthode MSA présentée en fin de section) donc nous présentons ci-après cet algorithme d'alignement global.

2.3.2. Algorithme Needleman-Wunsch

L'algorithme Needleman-Wunsch est un algorithme d'alignement global [10, 9]. Soient deux séquences S et T de taille n et m, Needleman-Wunsch utilise des notions de programmation dynamique pour trouver l'alignement global optimal entre S et T en $O(nm)$. Soit $V(i,j)$ le score de l'alignement optimal entre $S[1\dots i]$ et $T[1\dots j]$. Soit $\delta(a,b)$ le score de similarité entre deux lettres a et b. La méthode définie une formule récursive pour $V(i,j)$ selon deux cas de figure : (1) soit $i=0$ ou $j=0$; (2) soit $i>0$ et $j>0$. Dans le premier cas, on aligne une séquence avec une séquence vide (nous avons donc des insertions ou suppressions) et l'on a :

$$\begin{aligned} V(0,0) &= 0 \\ V(0,j) &= V(0,j-1) + \delta(\_, T[j]) \quad \text{Insère } j \text{ fois} \\ V(i,0) &= V(i-1,0) + \delta(S[i], \_) \quad \text{Supprime } i \text{ fois} \end{aligned}$$

Dans le deuxième cas, lorsque $i>0$ et $j>0$, dans le meilleur alignement entre $S[1\dots i]$ et $T[1\dots j]$, la dernière paire de lettres alignées devrait être soit une correspondance entre les lettres, une suppression ou une insertion de lettres. Le score optimal correspondra à la valeur maximum entre ces trois cas, on a :

$$V(i,j) = \max \begin{cases} V(i-1,j-1) + \delta(S[i], T[j]) & \text{correspondance} \\ V(i-1,j) + \delta(S[i], \_) & \text{suppression} \\ V(i,j-1) + \delta(\_, T[j]) & \text{insertion} \end{cases}$$

Le score de l'alignement optimal de $S[1\dots n]$ et $T[1\dots m]$ est $V(n,m)$. Pour obtenir ce score, les cases d'une table de taille n par m sont remplies en utilisant les équations récursives présentées ci-dessus. Le score de l'alignement optimal est indiqué dans la dernière case de la table. Pour retrouver l'alignement optimal, pour chaque case, une flèche noire retrace le chemin emprunté pour arriver à la dernière case de la table. Si la flèche est diagonale, les deux lettres correspondantes sont alignées. Si la flèche est horizontale ou verticale alors il y a une suppression ou une insertion (respectivement) dans l'alignement. La version classique de Needleman-Wunsch nécessite de remplir une table de taille n par m donc l'algorithme est en $O(nm)$ en temps et en espace (étant donné qu'une case se remplie en $O(1)$). D'autres versions ont été proposées dans la littérature pour améliorer la complexité temporelle ou spatiale de la version de base [12, 13] (la meilleure méthode actuelle en temps est en $O(nm/\log n)$ et la meilleure en espace est en $O(m+n)$).

Il est possible d'utiliser les méthodes pair-à-pair sur un ensemble de plus de trois séquences pour trouver des similitudes entre ces séquences. Cependant, les méthodes pair-à-pair ne peuvent pas identifier des sections conservées entre toutes les séquences [9]. De plus, la plupart des formulations du problème d'alignement de séquences multiples conduisent à des problèmes d'optimisation combinatoire NP-complets [14, 15, 16]. Ainsi, des méthodes heuristiques ont été développées pour traiter le problème. Il convient donc d'utiliser des méthodes heuristiques et dans notre cas nous présentons `ClustalW` et `MUSCLE`.

2.3.3. `ClustalW` et `MUSCLE`

Bien que les méthodes heuristiques ne peuvent pas garantir l'optimalité de leurs solutions, elles sont généralement en mesure de fournir de bons alignements multiples. Les heuristiques proposées dans la littérature peuvent être globalement classées en deux approches [9] :

1. L'**approche progressive** (e.g. `ClustalW`) consiste à utiliser des alignements pair-à-pair pour guider progressivement l'alignement multiple. Cette approche consiste à d'abord aligner les deux séquences les plus proches ; puis les séquences suivantes les plus étroitement liées sont progressivement alignées jusqu'à ce que toutes les séquences soient alignées. En général, une méthode d'alignement progressive comprend trois étapes : (1) le calcul des distances pair-à-pair pour toutes les paires de séquences ; (2) la construction d'un arbre directeur garantissant que des séquences similaires sont plus proches dans l'arbre et (3) l'alignement des séquences, une par une, selon l'arbre.
2. L'**approche itérative** (e.g. `MUSCLE`) consiste généralement à obtenir un alignement multiple puis à l'améliorer itérativement. Étant donné que les méthodes progressives ne réalignent pas les séquences, si l'alignement initial est mauvais alors l'alignement de séquences multiples peut être de mauvaise qualité. Cela signifie également que ces méthodes sont sensibles à la distribution des séquences dans le jeu de séquences. L'approche itérative vise à éviter ces limitations inhérente aux méthodes progressives.

Soit un ensemble de séquences $S = \{S_1, S_2, \ldots, S_k\}$ chacune de taille n. `ClustalW` est une méthode progressive d'alignement multiple qui, dans un premier temps, utilise l'algorithme de Needleman-Wunsch pour calculer l'alignement global optimal entre chaque paire de séquences (S_i, S_j) (avec $i, j \in \{1, 2, \ldots, k\}$). Ensuite, la distance entre S_i et S_j est définie comme $1 - \frac{y}{x}$ où x et y sont, respectivement, le nombre de positions ne correspondant pas à des écarts et le nombre de positions identiques dans l'alignement entre S_i et S_j. Cette distance pair-à-pair est calculée pour chaque paires de séquences pour obtenir une matrice des distances (étape 1). L'arbre directeur est ensuite générer à partir de cette matrice des distances (étape 2) et c'est en s'aidant de cet arbre que l'algorithme effectue une série d'alignements pour aligner des groupes de séquences de plus en plus grands (étape 3).

Après ces étapes, `ClustalW` fournie l'alignement multiple final. La complexité en temps de l'étape 1 est en $O(k^2n^2)$ étant donné que k^2 alignements globaux sont effectués. La complexité de l'étape 2 dépend de la méthode utilisée pour générer l'arbre. En utilisant l'algorithme "neighbour joining" [17] la complexité de l'étape 2 est en $O(k^3)$. L'étape 3 est en $O(kn^2)$ étant donné que l'arbre directeur a, au plus, k noeuds. Ainsi, la complexité de la version classique de `ClustalW` est en $O(k^2n^2 + k^3)$. Cependant, après certaines améliorations (e.g. utiliser une méthode heuristique pour obtenir la matrice des distances) [18, 19], la complexité de la version 2.0 de `ClustalW` est en $O(k^2)$.

`MUSCLE` est une méthode itérative d'alignement multiple fonctionnant en trois étapes : (1) un alignement multiple initial est générer avec une méthode progressive (e.g. ClustalW) ; (2) la méthode progressive est réutilisée pour améliorer l'alignement et enfin (3) des ajustements sont faits pour continuer d'améliorer l'alignement. De plus, `MUSCLE` utilise quelques

stratégies [20, 9] pour améliorer la précision de la méthode progressive utilisée à l'étape 1 et 2 mais nous n'aborderons pas ces modifications ici.

Pour conclure, la première méthode de fusion des *baitModels* (que nous présentons dans la section suivante) peut utiliser `ClustalW` ou `MUSCLE` pour les aligner. Ces *baitModels* sont l'interprétation des *hitModifieds* générés par `SpecGlob` à partir de PSM fournies par `SpecOMS`. L'objectif de la première méthode est d'aligner les *baitModels*, en tenant compte des écarts de masses éventuels, afin de trouver une séquence considérant les informations pertinentes portées par chaque *baitModels*. La séquence ainsi obtenue, que nous appelons *baitFusion*, est censée représenter le même peptide que les *baitModels*. En d'autres mots, *baitFusion* est supposée représenter :

- les sections de résidus conservées entre tous les *baitModels*, et
- les sections ambiguës (i.e. les décalages de masse) des *baitModels* qui sont potentiellement sous forme de séquences de résidus dans d'autres *baitModels*

Nous présentons dans la section suivante nos deux méthodes de fusion des *baitModels*. Les algorithmes MSA sont utilisés par la première méthode et la deuxième utilise une autre approche sans recours à ces algorithmes.

3. Méthodes de fusion

Les baitModels peuvent représenter une même séquence d'acides aminés (i.e. un peptide), potentiellement à quelques erreurs près. Il conviendrait, sur la base des baitModels fournis, de reconstruire le peptide en utilisant ou en ignorant les informations qu'ils portent. Les deux méthodes de fusion des *baitModels* présentées dans cette section prennent en entrée :

- une table de masses contenant les masses de toutes les séquences d'au plus n résidus (n étant un nombre arbitraire que nous fixons à 8)
- un fichier de statistiques contenant le nombre de *baits* à traiter, les *baits* (séquences) et pour chaque *bait* :
  - le nombre de *baitModels* qui lui sont associés et les *baitModels* (séquences).
  - la moyenne et l'écart-type des masses des *baitModels*.
  - le SPC, le score `SpecGlob`, la masse, la longueur de la sous-séquence de résidus la plus longue (notée LS [1]) et le nombre de décalages de masse de chaque *baitModel*.
  - le nombre de décalages où la masse du décalage correspond à une séquence unique d'au moins un résidu (nous notons GSC [2]) ; le nombre de décalages où la masse du décalage correspond à plusieurs séquences d'au moins un résidu (GMC [3]), et le nombre de décalages où la masse du décalage ne correspond à aucune séquence connue de moins de n résidus (GUM [4]).

1. LS = "*Longest Stretch*"
2. GSC = "*Gaps corresponding to Single amino acids Combination*"
3. GMC = "*Gaps corresponding to Multiple amino acids Combinations*"
4. GUM = "*Gaps corresponding to Unknown amino acids Combination*"

Certaines des données présentes dans le fichier de statistiques ne sont utilisées que pour vérifier que les *baitModels* ont bien la même masse (e.g. moyenne et écart-type des masses des *baitModels*). La première méthode de fusion des *baitModels*, nommée `alignBaitFusion`, utilise les algorithmes MSA présentés dans la section 2. La deuxième méthode de fusion, que nous appelons `linearBaitFusion`, utilise des curseurs sur chaque *baitModels* et ces curseurs sont avancés en fonction de la concordance entre tous les acides aminés.

3.1. Première méthode : `alignBaitFusion`

Le principe de la première méthode est de pouvoir utiliser des algorithmes de la littérature (e.g. ClustalW, MUSCLE) pour aligner les *baitModels* et déterminer une séquence consensus. Plus précisément, les décalages de masse des baitModels sont remplacées par des écarts de taille indéfinie (représentés par des "−") avant d'aligner les *baitModels* :

- les décalages de masse de type GMC trouvés dans la table des masses sont remplacés par k tirets ("−") où k est la moyenne des longueurs des combinaisons correspondant à la masse du décalage.
- les décalages de masse n'ayant aucune correspondance dans la table des masses (de type GUM) sont remplacés par l tirets où M est la masse du décalage, $m(G)$ est la masse de l'acide aminé G (Glycine, acide aminé le moins lourd) et $l = \left\lfloor \frac{M}{m(G)} \right\rfloor$.
- les séquences obtenues en remplaçant les masses par des tirets sont fournies à un algorithme d'alignement de séquences multiples (e.g. ClustalW, MUSCLE)
- les séquences alignées sont récupérées en sortie de l'algorithme MSA et une procédure d'élection est effectuée. Cette procédure lit le premier caractère de chaque séquence et choisie un résidu à la majorité (i.e. le caractère le plus présent parmi l'ensemble des caractères lus). Ensuite, la procédure lit les caractères suivants un par un pour élire le reste des résidus à la majorité et l'ensemble des résidus élus constitue *baitFusion* (la fusion des *baitModels*). Si le caractère "−" est élu il est ignoré et n'est pas inséré dans *baitFusion*. L'algorithme s'arrête lorsque le résidu K (ou R) est ajouté à *baitFusion* ou lorsque tous les caractères des séquences sont lus.

Dans le cas où deux résidus sont ex aequo lors de la procédure d'élection, un score noté $score_B$ est utilisé pour les départager et élire un candidat. Ce score est définie comme suit :

$$score_B = \sum_{b \in B} e^{\left(\frac{\mathrm{LS}_b}{10+10*\mathrm{GSC}_b+100*\mathrm{GMC}_b+1000*\mathrm{GUM}_b}\right)}$$

où B est l'ensemble des *baitModels* votant pour un résidu donné. L'idée derrière ce score est de favoriser les *baitModels* ayant le plus de décalages de masse non ambigus (du moins ambigu au plus ambigu nous avons les décalages GSC, GMC puis GUM) et les plus longues sous-séquences d'acides aminés. Par exemple, soit deux *baitModels* b_1 et b_2 associés à un *bait* et où b_1 et b_2 ont tous les deux un longest stretch de 5 résidus. De plus, b_1 a deux décalages (i.e. masse entre crochets) de type GUM et vote pour l'acide aminé I tandis que b_2 a un décalage de type GSC et propose le résidu K. Dans cet exemple, $score_{\{b_1\}} = 1.00$ alors que $score_{\{b_2\}} = 1.28$ donc c'est le résidu K proposé par b_2 qui est élu parmi les candidats I et K.

D'autre part, dans un effort d'éliminer le plus possible les décalages de type GUM (difficile ou impossible à interpréter *a priori*) une procédure de simplification des *baitModels* peut être utilisée si l'utilisateur le souhaite. Soit d_{GUM} la masse d'un décalage de type GUM et d_{adj} la masse d'un décalage suivant (ou précédant) le décalage de type GUM. La procédure de simplification identifie les décalages de type GUM et remplace ce décalage, le décalage le suivant (ou le précédant) et les résidus entre les deux décalages par un décalage de $d_{\text{GUM}}+d_{\text{adj}}+m_{\text{mid}}$ Da où m_{mid} correspond à la masse des résidus entre les deux décalages. Par exemple, considérons le *bait* GTFQIVYK avec deux *baitModels* G[376.17]IVYK et G[89.09]YVI[287.09]K. Le décalage de 89.09 Da du deuxième *baitModel* ne correspond à aucune séquence d'acides aminés dans la table de masse. Nous remplaçons donc [89.09]YVI[287.09] par un décalage de $89.09 + m(Y) + m(V) + m(I) + 287.09$ Da c'est-à-dire que le deuxième *baitModel* sera remplacé par G[751.38]K.

Dans l'algorithme 1, la fonction `empty_string` renvoie une chaîne de caractères vide et la fonction `length` renvoie la longueur de la chaîne de caractères passée en paramètre. La fonction `getBaitModelScore` renvoie le $score_B$ associé à un ensemble de *baitModels* tandis que la fonction `selectCandidateWithBestScore` sélectionne le candidat ayant le plus de vote (ou le meilleur $score_B$ en cas d'ex aequo) ou "$-$" s'il est impossible d'élire un candidat. Enfin, les fonctions `convertBaitModels` et `runMSA` permettent de remplacer les masses par des tirets comme décrit précédemment et, respectivement, d'aligner les séquences obtenues en utilisant un algorithme d'alignement de séquences multiples. Le format utilisé pour fournir les séquences aux algorithmes MSA est le format FASTA [21], c'est aussi le format des séquences de sortie. Les *baitModels* sont stockés dans des tableaux et pour compter le nombre de *baitModels* ayant voté pour un résidu nous utilisons une table de hachage qui à un résidu associe un compteur. De la même façon, le $score_B$ attribué à un résidu est stocké dans une table de hachage (représentée par des $\langle\rangle$).

Le principal défaut d'`alignBaitFusion` est l'usage des tirets (une information imprécise) pour remplacer les décalages de masse (une information précise). Il conviendrait d'utiliser les décalages sous leurs forme d'origine et d'inclure l'information inhérente à ces décalages. De plus, il faudrait un mécanisme pour donner plus d'importance aux *baitModels* qui ont tendance à s'accorder sur les résidus proposés lors de la procédure d'élection. Ce mécanisme serait aussi responsable d'ignorer les *baitModels* en désaccord avec la majorité. La méthode `linearBaitFusion` que nous proposons prend en compte ces remarques.

3.2. Deuxième méthode : `linearBaitFusion`

L'idée de la deuxième méthode est de placer un curseur sur chaque *baitModel*. Ces curseurs sont des têtes de lecture pouvant lire un seul acide aminé à la fois et se déplaçant à des rythmes différents selon un critère de validité du *baitModel*. L'algorithme est itératif et se décline en deux phases à chaque itération :

1. L'acide aminé sous chaque curseur est lu et est considéré comme un candidat. Une élection à la majorité est alors effectuée pour choisir un seul candidat (e.g. si 4 curseurs ont lu le résidu Y et 2 autres ont lu le résidu G alors l'élu sera Y). Le $score_B$ de la

Algorithme 1 Algorithme première méthode

```
fonction alignBaitFusion(baitModels) :
    n ← length(baitModels)
    candidate ← '−'
    indices[n] ← {0}
    keepgoing ← True
    baitFusion ← empty_string()

    [baitModels ← simplify(baitModels)]                          ▷ Optionnel
    dashedSequences ← convertBaitModels(baitModels)
    outputSequences ← runMSA(dashedSequences)
    Tant que candidate ≠ K et candidate ≠ R et keepgoing, faire :
        candidate ← '−'
        scoresB ← ⟨⟩
        candidates ← ⟨⟩

        Pour i de 1 à n, faire :
            c ← '−'
            Si 0 ⩽ indices[i] < length(baitModels[i]) alors
                c ← baitModels[i][indices[i]]
                indices[i] ← indices[i]+1
            Fin si
            Si c ≠ alors '−'
                Si c ∉ candidates alors
                    candidates[c] ← 0
                    scoresB[c] ← 0
                Fin si
                candidates[c] ← candidates[c] + 1
                scoresB[c] ← scoresB[c] + getBaitModelScore(baitModels[i])
            Fin si
        Fin pour

        candidate ← selectCandidateWithBestScore(candidates, scoresB)
        Si candidate ≠ alors '−'
            baitFusion ← baitFusion + candidate
        Fin si
    Fin tant que

    renvoyer baitFusion
Fin fonction
```

première méthode est aussi utilisée dans cette méthode mais un autre score, que nous notons $score_A$, est d'abord utilisé (voir suite). Il est aussi possible pour un *baitModel* de proposer un décalage de masse comme candidat au lieu d'un résidu, la masse du candidat élu sera alors soustraite de la masse du décalage si possible.

2. Une fois l'élection terminée, les *baitModels* ayant voté pour le candidat élu sont considérés comme **valides**. Les *baitModels* ayant voté pour un autre acide aminé (différent du candidat élu) sont considérés comme étant en **probation**, si lors d'une prochaine élection ils proposent un candidat ne correspondant pas au candidat élu alors ils seront considérés comme **invalides**. D'autre part, pour chaque décalage de masse proposé par un *baitModel*, la masse du candidat élu est soustraite à la masse du décalage. Le *baitModel* proposera le décalage avec la nouvelle masse obtenue au prochain tour si cette masse est non négligeable (> 1 Da). Si la masse du décalage est négligeable (entre -1 et 1 Da), le décalage n'est plus pris en compte dans les prochaines itérations et c'est les résidus qui le suivent qui seront proposés par le *baitModel*. Cependant, si la masse du décalage est négative (< -1 Da) alors le *baitModel* proposant ce décalage est dit **définitivement invalide**. Enfin, si la masse du décalage correspond à la masse d'un résidu c'est le résidu en question qui sera proposé.

L'étape 2 de cette méthode est l'étape de validation des *baitModels*. Après cette étape, les curseurs des *baitModels* n'étant pas invalides sont avancés (de la gauche vers la droite). La concaténation des résidus élus est appelée *baitFusion* (la fusion des *baitModels*). L'algorithme s'arrête lorsque le résidu K (ou R) est ajouté à *baitFusion*, lorsqu'aucun *baitModel* ne propose de candidat ou lorsque tous les caractères des *baitModels* sont lus.

En plus de ces deux étapes, certains mécanismes ont été ajoutés pour ne pas trop pénaliser les *baitModels* ou pour gérer les situations d'ex aequo (voir illustrations Annexe B.2) :

- Les *baitModels* sont pondérés : un *baitModel* valide = 4, un *baitModel* en probation = 1 et un *baitModel* invalide = 0. Lors d'un ex aequo entre plusieurs candidats, un premier score, $score_A$, est utilisé pour départager les candidats. Ce score correspond à la somme des pondérations attribuées aux *baitModels* votant pour un candidat donné $score_A(B) = \sum_{b \in B} p_b$ où B est l'ensemble des *baitModels* votant pour un résidu donné et p_b la pondération attribuée au *baitModel* b. L'idée est de favoriser les candidats proposés par les *baitModels* de meilleure pondération. Dans le cas où les scores $score_A$ sont égaux pour plusieurs ensemble de *baitModels* et où aucun candidat ne peut être élu, le $score_B$ présenté pour `alignBaitFusion` est utilisé. Si le $score_B$ ne peut pas départager les candidats alors l'algorithme est interrompu.
- Une stratégie de réinsertion (ou "*comeback*") est utilisée pour poursuivre l'utilisation de *baitModels* précédemment invalidés (sauf ceux définitivement invalidés). Lors de la phase de validation, si le curseur d'un *baitModel* invalide est sur un résidu correspondant à l'élu actuel alors il est placé en probation. L'intérêt est de réintégrer des *baitModels* précédemment mis à l'écart parce qu'ils sont susceptibles de contenir des résidus pertinents dont les positions ne correspondent pas avec les positions des résidus de *baitModels* valides ou en probation.

La méthode en deux phases munie des stratégies de pondération et de réinsertion constitue la première version de la méthode `linearBaitFusion`. La procédure de simplification de la méthode `alignBaitFusion` peut être utilisée en amont de la méthode `linearBaitFusion` pour tenter de simplifier les décalages de type GUM. Dans l'algorithme 2, la fonction `empty_string` renvoie une chaîne de caractères vide et la fonction `length` renvoie la longueur de la chaîne de caractères passée en paramètre. Le prédicat `isAminoAcid` renvoie un booléen précisant si l'élément passé en paramètre est un acide aminé ou non. De la même façon, le prédicat `isMass` renvoie un booléen précisant si l'élément passé en paramètre est un décalage de masse ou non. `readMass` renvoie la masse se situant entre crochets si le curseur est placé sur un décalage de masse. Les fonctions `getAminoAcidMass` et `absoluteValue` permettent de retourner la masse d'un acide aminé et, respectivement, la valeur absolue du nombre passé en paramètre. Enfin, la fonction `getBaitModelScore` renvoie le $score_B$ associé à un ensemble de *baitModels* tandis que la fonction `selectCandidateWithBestScore` sélectionne le candidat ayant : le plus de vote, le meilleur $score_A$ en cas d'ex aequo, le meilleur $score_B$ s'il y a ex aequo avec le $score_A$ ou "_" s'il est impossible d'élire un candidat.

Toutefois, cette version de `linearBaitFusion` a deux inconvénients. Le premier inconvénient est l'éventuelle absence d'information dans *baitFusion* lorsque l'algorithme est interrompu parce qu'aucun candidat n'est proposé. En effet, dans ce cas de figure, la séquence obtenue jusqu'à l'arrêt est retournée en l'état alors qu'il est possible d'utiliser la masse de la séquence obtenue et la masse des *baitModels* (supposée être la même pour tous les *baitModels*) pour déterminer la masse des résidus qu'il reste à trouver. Par exemple, considérons le *bait* GTFQIVYK avec deux *baitModels* G[376.17]IVYK et G[751.38]K (version simplifiée de G[89.09]YVI[287.09]K). Les deux *baitModels* propose le résidu G donc G est élu et ajouté à *baitFusion*. Lors de la deuxième itération, les curseurs des deux *baitModels* sont sur des décalages de masse et aucune de ces masses ne correspond à la masse d'un acide aminé donc aucun résidu n'est proposé. L'algorithme est alors interrompu et la séquence obtenue est G (de masse 57.02 Da). Les *baitModels* ont une masse de 936.48 Da donc la masse des résidus qu'il reste à trouver est 936.48 - 57.02 soit 879.46 Da. Ainsi, au lieu de retourner G, nous pouvons retourner la séquence G[879.46] pour indiquer qu'une séquence de résidus se trouvant après G et dont la masse correspond à 879.46 Da n'a pas pu être reconstituée.
Le deuxième inconvénient est une conséquence direct du premier. Lorsque l'algorithme est interrompu parce qu'aucun candidat n'est proposé, il peut s'agir d'une situation où tous les *baitModels* propose des décalages de masse et où aucune de ces masses ne correspond à la masse d'un et un seul acide aminé. Il conviendrait alors d'entamer une résolution utilisant des curseurs placés à la fin de chaque *baitModel* et se déplaçant de la droite vers la gauche. De cette façon, nous pouvons reconstitué le début et la fin des *baitModels*. Ensuite, nous pouvons utiliser les deux séquences obtenues et la masse des *baitModels* pour fusionner les deux séquences. Reprenons l'exemple du paragraphe précédent, la séquence obtenue de la gauche vers la droite est G. Ensuite, un parcours de la droite vers la gauche est effectué et les résidus K, Y, V et I sont reconstitués. Nous obtenons la séquence KYVI c'est-à-dire IVYK en conservant le sens de lecture de la gauche vers la droite. La fusion des séquences G et IVYK (sans oublier la masse des résidus non retrouvés) nous donne G[376.17]IVYK.

Algorithme 2 Algorithme de la deuxième méthode

```
fonction linearBaitFusion(baitModels) :
    n ← length(baitModels)
    candidate ← _
    keepgoing ← True
    indices[n] ← {0}                                  ▷ Initialisation des curseurs
    masses[n] ← {0.0}
    validation[n] ← {4}
    baitFusion ← empty_string()
    Tant que candidate ≠ K et candidate ≠ R et keepgoing, faire :
        candidate ← DoElection(n, baitModels, indices, masses, validation)
        Si pas isAminoAcid(candidate) alors
            keepgoing ← False
            sortir de la boucle
        Sinon
            baitFusion ← baitFusion + candidate
        Fin si

        Pour i de 1 à n, faire :          ▷ Étape de validation des baitModels
            Si 0 ⩽ indices[i] < length(baitModels[i]) alors
                c ← baitModels[i][indices[i]]
                Si c = candidate et absoluteValue(masses[i]) ⩽ 1 alors
                    Si validation[i] = 0 alors
                        validation[i] ← 1                          ▷ Comeback
                    Fin si
                Sinon
                    Si masses[i] ⩾ 1 alors
                        Si masses[i] − getAminoAcidMass(candidate) ⩾ −1 alors
                            masses[i] ← masses[i] − getAminoAcidMass(candidate)
                        Sinon
                            indices[i] ← −1
                            validation[i] ← 0
                        Fin si
                    Sinon, si isAminoAcid(c) et validation[i] = 4
                        validation[i] = 1
                    Sinon
                        validation[i] = 0
                    Fin si
                Fin si
                indices[i] ← indices[i] + 1
            Fin si
        Fin pour
    Fin tant que
    renvoyer baitFusion
Fin fonction
```

Algorithme 3 Fonction `DoElection`

```
fonction DoElection(n, baitModels, indices, masses, validation) :
    candidate ← _
    scoresA ← ⟨⟩
    scoresB ← ⟨⟩
    candidates ← ⟨⟩

    Pour i de 1 à n, faire :
        c ← _
        Si 0 ⩽ indices[i] < length(baitModels[i]) alors
            c ← baitModels[i][indices[i]]
            Si isMass(c) alors
                masses[i] ← masses[i] + readMass(c)
            Fin si

            Si masses[i] < −1.0 alors
                validation[i] ← 0
                sauter l'itération actuelle
            Sinon, si absoluteValue(masses[i]) ⩾ 1
                Si isAminoAcid(masses[i]) alors
                    c ← getAminoAcidWithMass(masses[i])
                Fin si
            Sinon
                masses[i] ← 0.0
            Fin si
        Fin si

        Si c ≠ _ alors
            Si c ∉ candidates alors
                candidates[c] ← 0
                scoresA[c] ← 0
                scoresB[c] ← 0
            Fin si
            candidates[c] ← candidates[c] + 1
            scoresA[c] ← scoresA[c]+validation[i]
            scoresB[c] ← scoresB[c]+validation[i] ∗ getBaitModelScore(baitModels[i])
        Fin si
    Fin pour

    candidate ← selectCandidateWithBestScore(candidates, scoresA, scoresB)
    renvoyer candidate
Fin fonction
```

En conséquence, nous avons développé une deuxième version de `linearBaitFusion` prenant en compte ces inconvénients. Dans cette version, si aucun candidat ne peut être élu, la séquence (notée s_1) trouvée jusque là est stockée en mémoire, les curseurs sont placés à la fin de leur *baitModel* respectif et les *baitModels* sont reconsidérés comme étant valides. Les mêmes mécanismes et fonctions sont utilisés mais cette fois-ci les curseurs se déplacent de la droite vers la gauche. La séquence s_2 ainsi obtenue est inversée et stockée en mémoire. Ensuite, la méthode utilise la somme des masses de s_1 et s_2 et l'utilise en conjonction avec la masse des *baitModels* pour déterminer *baitFusion* (fonction `getSequence` dans l'algorithme 4). La fonction `prepareForReversal` réinitialise les variables comme décrit ci-dessus pour passer à la résolution de la droite vers la gauche. La fonction `advanceCursor` avance les curseurs dans la direction de résolution.

Pour être plus précis, la fonction `getSequence` vérifie si la résolution de la droite vers la gauche a été activée : si non alors la fonction renvoie s_1 et, si oui, la somme des masses de s_1 et s_2 (notée m_s) est comparée avec la masse des *baitModels* (notée m_B). Si $m_s = m_B$ (à 1 Da près) alors la fonction renvoie la concaténation de s_1 et s_2. Sinon, si $m_s > m_B$ alors au moins un résidu est de trop donc la fonction retire le résidu au début de s_2 jusqu'à ce que $m_s = m_B$ et renvoie la concaténation de s_1 et s_2. Enfin, si $m_s < m_B$ alors certains résidus n'ont pas pu être reconstitués, la fonction calcule la masse de la séquence non reconstituée (c'est-à-dire $m_B - m_s$) et essaie de trouver cette masse dans la table des masses :

- si le décalage de masse est de type GSC et que la masse correspond à celle d'un seul résidu ou d'une séquence unique de résidu répétés (e.g. G, KKK) alors la fonction renvoie la concaténation de s_1, de l'interprétation du décalage (i.e. résidu ou séquence de résidus répétés) et de s_2.
- si le décalage de masse est de type GSC mais que la masse correspond à une séquence de résidus différents alors nous savons quels résidus sont présents dans *baitFusion* mais nous ne savons pas dans quel ordre les reconstitués. La fonction renvoie la concaténation de s_1, la séquence de résidus entre accolades ("{}", pour indiquer que l'ordre des résidus n'est pas connue) et s_2.
- tout autre type de décalage est ambigu donc la masse du décalage est laissée telle quelle et la fonction renvoie la concaténation de s_1, la masse du décalage entre crochets et s_2.

De cette manière, la deuxième version de `linearBaitFusion` utilise autant que possible les informations fournies par les *baitModels*. L'avantage majeur des deux versions de la deuxième méthode par rapport à `alignBaitFusion` est la simplicité des mécanismes utilisés et l'absence d'algorithmes MSA (i.e. aucune configuration de ces algorithmes MSA n'est requise). D'un point de vue théorique, nous nous attendons à ce que `linearBaitFusionV2` (deuxième version de `linearBaitFusion`) soit plus efficace pour reconstruire la séquence peptidique des *baits* étant donné que c'est la méthode utilisant le plus les informations portées par les *baitModels*. Dans la section suivante, nous discutons du paramétrage de `alignBaitFusion`, `linearBaitFusion` et `linearBaitFusionV2`. De plus, nous présentons les résultats numériques obtenus avec ces méthodes sur des *baits* issus du protéome humain.

Algorithme 4 Algorithme de la deuxième méthode (2e version)

```
fonction linearBaitFusionV2(baitModels) :
    n ← length(baitModels)
    candidate ← _
    reversed, keepgoing ← False, True
    indices[n], masses[n] ← {0}, {0.0}                    ▷ Initialisation des curseurs
    validation[n] ← {4}
    s_1, s ← empty_string(), empty_string()
    Tant que candidate ≠ K et candidate ≠ R et keepgoing, faire :
        candidate ← DoElection(n, baitModels, indices, masses, validation)
        Si pas reversed et pas isAminoAcid(candidate) alors
            reversed ← True
            s_1 ← copy(s)
            prepareForReversal(baitModels, indices, masses, validation, s)
        Sinon, si pas isAminoAcid(candidate)
            keepgoing ← False
            sortir de la boucle
        Sinon
            s ← s + candidate
        Fin si
        Pour i de 1 à n, faire :                          ▷ Étape de validation des baitModels
            Si 0 ⩽ indices[i] < length(baitModels[i]) alors
                c ← baitModels[i][indices[i]]
                Si c = candidate et absoluteValue(masses[i]) ⩽ 1 alors
                    Si validation[i] = 0 alors
                        validation[i] ← 1                 ▷ Comeback
                    Fin si
                Sinon
                    Si masses[i] ⩾ 1 alors
                        Si masses[i] − getAminoAcidMass(candidate) ⩾ −1 alors
                            masses[i] ← masses[i] − getAminoAcidMass(candidate)
                        Sinon
                            indices[i] ← −1
                            validation[i] ← 0
                        Fin si
                    Sinon, si isAminoAcid(c) et validation[i] = 4
                        validation[i] = 1
                    Sinon
                        validation[i] = 0
                    Fin si
                Fin si
                advanceCursor(indices, i, reversed)
            Fin si
        Fin pour
    Fin tant que
    baitFusion ← getSequence(s_1, s, reversed, baitModels)
    renvoyer baitFusion
Fin fonction
```

Algorithme 5 Procédure `prepareForReversal`

procédure prepareForReversal(baitModels, indices, masses, validation, s) :
  n ← `length`(baitModels)
  masses[n] ← {0.0}
  validation[n] ← {4}
  s ← `empty_string`()
  Pour i **de** 1 **à** n, **faire** :
    indices[i] ← `length`(baitModels[i]) − 1
  Fin pour
Fin procédure

Algorithme 6 Procédure `advanceCursor`

procédure advanceCursor(indices, i, reversed) :
  Si reversed **alors**
    indices[i] ← indices[i] − 1
  Sinon
    indices[i] ← indices[i] + 1
  Fin si
Fin procédure

4. Expérimentation numérique

Les *baits* sur lesquels nous avons testé nos méthodes sont des spectres théoriques. En effet, pendant sa thèse, Lysiak a fait le choix de mener les identifications de spectres par `SpecOMS` avec des spectres théoriques uniquement. En d'autres mots, un *bait* est en réalité un spectre théorique (représentant un peptide) qui joue le rôle de spectre expérimental dans le travail d'identification effectué par `SpecOMS`. Dans cette section, nous discutons des données que nous utilisons, des paramètres des méthodes présentées jusqu'ici et des résultats obtenus avec ces méthodes.

4.1. Données d'entrée

Les spectres théoriques ont été générés à partir des peptides issus du protéome humain ([22], `http://ftp.ensembl.org/pub/release-99/fasta/homo_sapiens/pep/`). Ce protéome contient 110 210 protéines fournies en entrée à SpecOMS. Afin de prendre en compte les contaminants, la base de donnée dédiée cRAP ("*common Repository of Adventitious Proteins*", `http://ftp.thegpm.org/fasta/cRAP/`), de 116 protéines, a été donnée en entrée à `SpecOMS`. Le logiciel est paramétré de façon à digérer les protéines *in silico*, à la manière de la trypsine. `SpecOMS` sélectionne 572 063 peptides de longueur comprise entre 7 et 30 résidus après digestion du protéome humain et de la base cRAP. Cet ensemble de peptides (transformés en spectres théoriques) est appelé *base target*. Un autre ensemble de peptides appelé *base decoy* est présenté dans la thèse de Lysiak mais il est ignorer dans le contexte de nos travaux. Les *hits* et les *baits* sont des instances de la base target.

Le paramètre t (seuil de SPC) de `SpecOMS` est fixé à 7 donc le logiciel extrait de `SpecTrees` toutes les paires de spectres avec un SPC d'au moins $t = 7$. Ce seuil semble pertinent car il ne doit pas être trop bas pour ne pas produire des PSM contenant peu d'information, qui de plus seront lents à produire et formeront un fichier trop volumineux [5]. Il ne doit pas être trop haut non plus puisque le SPC peut décroître très rapidement lorsque le nombre de modifications augmente ; or, nous souhaitons autoriser la présence de PSM modifiés. Après comparaison des spectres théoriques aux spectres expérimentaux, `SpecOMS` renvoie 2 532 091 PSM soient 453 961 *baits*. Le logiciel `SpecGlob` est ensuite utilisé pour générer les *hitModifieds* (et par extension, les *baitModels*) à partir des PSM.

Enfin, un fichier de statistiques (voir section 3) sur les 453 961 *baits*, obtenus avec `SpecGlob`, est générer à l'aide d'un script. De la même façon, un script est utilisé pour générer une table au format csv où la première colonne correspond à une masse (en Dalton) et les colonnes suivantes correspondent aux séquences de résidus (composés de 8 résidus maximum) ayant cette masse. Ce script stocke les séquences dans l'ordre lexicographique des résidus dans la table de masse (étant donné que l'ordre des résidus importe peu). Les méthodes présentées dans la section 3 utilisent le fichier de statistiques et la table de masse pour charger et fusionner les *baitModels*. Cependant, ces méthodes sont configurées pour ne considérer que les *baits* avec au moins deux *baitModels* différents. Ainsi, les méthodes sont exécutées sur 338 715 *baits* au lieu des 453 961 *baits* obtenus avec `SpecGlob`. Chaque *bait* considéré a entre 2 et 143 *baitModels*. De plus, 132 959 *baits* ont au moins un *baitModel* dont la séquence correspond exactement à celle du *bait* auquel il est associé, nous dirons dans ce cas que "le *bait* est inclus dans les *baitModels*". Enfin, les masses stockées dans la table ont au plus deux chiffres après la virgule. Lorsque nous recherchons la masse d'un décalage porté par un *baitModel* dans la table, il faut tenir compte des imprécisions inhérentes aux nombres décimaux. Nous expliquons ci-après comment nous avons paramétré nos méthodes pour tenir compte de ces imprécisions et pour obtenir des résultats corrects.

4.2. Paramétrage des méthodes

Les méthodes `alignBaitFusion`, `linearBaitFusion` et `linearBaitFusionV2` ont toutes trois paramètres dont le but est de minimiser les problèmes d'imprécisions. Ces paramètres ne sont pas explicités dans les algorithmes 1, 2 et 4 par soucis de simplicité. Ces trois paramètres sont :

- **trace** : un nombre décimal fixé à 1.0 par défaut et représentant une masse en Dalton. Ce paramètre est utilisé pour déterminer si un décalage de masse est négligeable (s'il est entre $-$trace et trace) ou non (décalage supérieur à trace ou inférieur à $-$trace). Par exemple, un décalage de masse de 0.5 Da sera considéré comme négligeable.
- **tolérance** : un nombre décimal fixé à 0.04 Da par défaut. Lorsque nous recherchons la masse d d'un décalage porté par un *baitModel* dans la table, nous recherchons cette masse dans l'intervalle suivant $[d - \text{tolerance}, d + \text{tolerance}]$. Ce paramètre est utilisé en conjonction avec le paramètre sensibilité.

- **sensibilité** : un nombre décimal fixé à 0.01 Da par défaut. Utilisé en conjonction avec tolérance, ce paramètre est le pas avec lequel l'intervalle $[d - \text{tolerance}, d + \text{tolerance}]$ est balayé. En d'autres termes, lors d'une recherche dans la table, nous vérifions si la masse d se trouve dans la table. Si elle n'est pas trouvée dans la table, une nouvelle recherche est effectuée avec les masses $d - \text{sensibilité}$ et $d + \text{sensibilité}$. Si aucune de ces masses n'est trouvée dans la table, une nouvelle recherche est effectuée avec les masses $d - 2 * \text{sensibilité}$ et $d + 2 * \text{sensibilité}$. Les recherches sont renouvelées ainsi de suite tant qu'aucune des masses proposées n'est retrouvée dans la table et tant que les masses proposées sont dans l'intervalle $[d - \text{tolerance}, d + \text{tolerance}]$. Par exemple, avec les valeurs par défaut des paramètres, si une recherche est effectuée pour un décalage de masse de 99.08 Da alors les masses suivantes sont successivement recherchées dans la table : 99.08, 99.07, 99.09, 99.06, 99.10, 99.05, 99.11, 99.04 et 99.12. Lorsqu'aucune de ces masses n'est retrouvée, la modification est alors classée comme **Rouge** (voir section 2.2) et le décalage associé est de type GUM (voir section 3).

Les pondérations des *baitModels* utilisées par `linearBaitFusionV2` et sa version classique peuvent aussi être considérées comme des paramètres. En effet, avec les pondérations par défaut (*baitModel* valide = 4, *baitModel* en probation = 1 et *baitModel* invalide = 0), les résidus proposés par les *baitModels* valides lors des élections auront 4 fois plus de voix en cas d'ex aequo (grâce au $score_A$). Les pondérations suivantes ont aussi été testées $\{\text{valide} = 2, \text{en probation} = 1, \text{invalide} = 0\}$ et $\{\text{valide} = 2, \text{en probation} = 1, \text{invalide} = -1\}$. Nous avons observé que la différence entre les résultats obtenus avec ces pondérations est négligeable mais la pondération $\{\text{valide} = 4, \text{en probation} = 1, \text{invalide} = -1\}$ est celle qui permet d'obtenir les meilleurs résultats. Ainsi, nous recommandons cette pondération.

De plus, nous avons vérifié expérimentalement qu'en donnant moins de voix aux résidus provenant de la masse d'un décalage, les méthodes donnent de meilleurs résultats. Par défaut, un résidu candidat provenant d'un décalage de masse obtient quatre fois moins de voix qu'un résidu retrouvé tel quel (i.e. sous la forme d'une lettre). L'ensemble des méthodes utilisent la procédure de simplification, présentée dans la section 3.1, par défaut.

Enfin, nous avons testé la méthode `alignBaitFusion` avec les algorithmes d'alignement de séquences multiples `ClustalW` et `MUSCLE` :

- `ClustalW` obtient des meilleurs résultats lorsque les paramètres suivant sont passés au logiciel "-QUICKTREE -MATRIX=ID -GAPOPEN=5 -GAPEXT=1 -NOWEIGHTS -CLUSTERING=UPGMA -NOHGAP". Les paramètres GAPOPEN et GAPEXT sont les pénalités pour insérer un nouveau décalage et pour étendre un décalage. MATRIX spécifie la matrice protéique que doit utiliser `ClustalW` pour comparer les résidus. Pour plus d'informations sur ces paramètres, nous invitons le lecteur a lire la documentation de `ClustalW` (`https://www.genome.jp/tools/clustalw/clustalw_help.html`)
- le logiciel `MUSCLE` est utilisé avec ces paramètres par défaut.

4.3. Résultats

Les méthodes présentées dans la section 3 ont toutes été testées sur une machine équipée d'un processeur Intel® Core™ i7-9750H de 9ème génération (6 coeurs, 12 threads, 2.6 GHz de base / 4.50 GHz maximum) sous Ubuntu 22.04.2 (LTS). La version de `ClustalW` utilisée est la version 2.0 tandis que la version de `MUSCLE` est la 3.8. Toutes les méthodes ont été implémentées en utilisant Python 3.10.6. Le temps moyen nécessaire pour lancer les méthodes sur l'ensemble des 338 715 *baits* est de 15 minutes pour la méthode `alignBaitFusion/ClustalW`, 20 minutes pour la méthode `alignBaitFusion/MUSCLE` et 5 minutes pour la méthode `linearBaitFusionV2` (et sa version classique). Les critères pour la comparaison entre les méthodes sont :

C1. le nombre de *baits* retrouvés.

C2. le nombre de séquences reconstituées sans décalages de masse ou séquence de résidus ambiguë ("Fully Reconstructed Sequences" ou FSR).

C3. le nombre de FSR correspondant à la séquence du *bait* associé.

C4. la longueur de la plus longue sous-séquence continue commune au *bait* et à *baitFusion*. C'est le critère utilisé pour déterminer le nombre de résidus que *baitFusion* possède en commun avec le *bait* associé.

	C1	**C2**	**C3**
alignBaitFusion/ClustalW	153 118 (45.21%)	338 715 (100.00%)	153 118 (45.21%)
alignBaitFusion/MUSCLE	114 103 (33.69%)	338 715 (100.00%)	114 103 (33.69%)
linearBaitFusion	195 408 (57.69%)	338 715 (100.00%)	195 408 (57.69%)
linearBaitFusionV2	203 321 (60.03%)	228 999 (67.61%)	203 321 (88.79%)

TABLE 1 – **Résultats obtenus avec nos méthodes.** Les critères de comparaison utilisés sont **C1**, **C2** et **C3** (voir ci-dessus). Ces chiffres sont, entre autres, obtenus à partir des tables A.2, A.3 et A.4.

La TABLE 1 indique que les méthodes `alignBaitFusion` et `linearBaitFusion` renvoie systématiquement des séquences sans décalages. Ces chiffres sont tout à fait compréhensibles étant donné que ces méthodes non pas été conçues pour utiliser les masses m_B des *baitModels* pour déterminer si la masse de *baitFusion* correspond bien à m_B. En d'autres mots, bien que `linearBaitFusionV2` renvoie dans certains cas des séquences portant un décalage ou une séquence de résidus ambiguë, les séquences retournées par `linearBaitFusionV2` sont riches en information. De plus, cette méthode est celle retrouvant le plus de *baits* (60.03% c'est-à-dire 203 321 *baits* retrouvés sur 338 715) et les résultats obtenus pour le critère **C3** montre que 88.79% des séquences reconstituées complètement (FSR) correspondent à la séquence *bait* qui leur est associé. En d'autres termes, seul 11.21% des FSR retrouvés par `linearBaitFusionV2` ne correspondent pas à leur *bait* contre 54.79% pour `alignBaitFusion/ClustalW`, 66.31% pour `alignBaitFusion/MUSCLE` et 42.31% pour `linearBaitFusion`. Enfin, les tables A.2, A.3 et A.4 montrent que seul 58 746 séquences ont moins de 7 résidus en commun avec leur *bait* pour `linearBaitFusionV2` contre 91 554 séquences pour `alignBaitFusion/ClustalW` et 137 736 pour `alignBaitFusion/MUSCLE`.

5. Conclusion

Dans ce rapport, nous avons présenté des méthodes permettant de fusionner des *baitModels*. Essentiellement, ces méthodes sont des algorithmes de reconstruction de séquences peptidiques et elles contribuent à l'identification de peptides portant des modifications chimiques à la fois multiples et inconnues. Notre meilleure méthode permet de reconstruire 60% des *baits* issus du protéome humain possédant au moins deux *baitModels* différents. De plus, cette méthode renvoie des séquences riche en information qui, même quand elles ne correspondent pas au peptide d'origine (i.e. le *bait*) décrivent le mieux que possible les *baitModels*.

Néanmoins, les spectres expérimentaux (i.e. les *baits*) que nous utilisons sont en réalité des spectres théoriques. La prochaine étape est donc d'appliquer nos méthodes sur des *baitModels* issus de spectres expérimentaux réels. Il faut toutefois garder en tête que, bien que les données utilisées ne comportent pas de bruits ou de pics manquants, tout le travail d'élaboration et d'amélioration de nos méthodes sur les données théoriques n'est pas incompatible avec l'adaptabilité des résultats aux spectres expérimentaux.

Enfin, lors de ce travail d'étude et de recherche (TER), j'ai pu consolider mes méthodes d'investigation et, au fil des réunions hebdomadaires avec mes encadrants, j'ai redécouvert la signification de la rigueur scientifique. La première difficulté de ce TER a été l'interdisciplinarité du sujet. Mais, contre toute attente, c'est aussi ce croisement entre différents domaines qui a réussit à attiser ma curiosité.

Remerciements

Je tiens à remercier mes encadrants, Guillaume FERTIN, Géraldine JEAN et Émile BENOIST, pour leur bienveillance, leurs remarques constructives et, bien évidemment, leur implication tout au long de ce TER. Merci encore pour votre réactivité lors de nos échanges par mail, pour les longues discussions très intéressantes et vos idées toujours très pertinentes.

Références

[1] V. C. Wasinger, S. J. Cordwell, A. Cerpa-Poljak, J. X. Yan, A. A. Gooley, M. R. Wilkins, M. W. Duncan, R. Harris, K. L. Williams, I. Humphery-Smith, Progress with gene-product mapping of the mollicutes: Mycoplasma genitalium, ELECTROPHORESIS 16 (1) (1995) 1090–1094. arXiv:https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/pdf/10.1002/elps.11501601185, doi:https://doi.org/10.1002/elps.11501601185.
URL https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/abs/10.1002/elps.11501601185

[2] G. S. Omenn, The hupo human proteome project (hpp), a global health research collaboration, Central Asian Journal of Global Health 1 (1) (Sep. 2012). doi:10.5195/cajgh.2012.37.
URL http://cajgh.pitt.edu/ojs/cajgh/article/view/37

[3] P. Conrotto, S. Souchelnytskyi, Proteomic approaches in biological and medical sciences : principles and applications., Experimental oncology 30 3 (2008) 171–80.

[4] R. Aebersold, M. Mann, Mass-spectrometric exploration of proteome structure and function, Nature 537 (7620) (2016) 347–355.

[5] A. Lysiak, Développement de méthodes informatiques pour l'évaluation et l'amélioration de l'identification par spectrométrie de masse des peptides modifiés, Theses, Nantes Université (Dec. 2022).
URL https://theses.hal.science/tel-03945760

[6] M. David, G. Fertin, H. Rogniaux, D. Tessier, Specoms : a full open modification search method performing all-to-all spectra comparisons within minutes, Journal of proteome research 16 (8) (2017) 3030–3038.

[7] V. O. Polyanovsky, M. A. Roytberg, V. G. Tumanyan, Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences, Algorithms for molecular biology 6 (1) (2011) 1–12.

[8] M. Brudno, S. Malde, A. Poliakov, C. B. Do, O. Couronne, I. Dubchak, S. Batzoglou, Glocal alignment : finding rearrangements during alignment, Bioinformatics 19 (suppl_1) (2003) i54–i62.

[9] W.-K. Sung, Algorithms in bioinformatics : A practical introduction, CRC Press, 2009.

[10] S. B. Needleman, C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, Journal of molecular biology 48 (3) (1970) 443–453.

[11] T. F. Smith, M. S. Waterman, et al., Identification of common molecular subsequences, Journal of molecular biology 147 (1) (1981) 195–197.

[12] W. J. Masek, M. S. Paterson, A faster algorithm computing string edit distances, Journal of Computer and System sciences 20 (1) (1980) 18–31.

[13] D. S. Hirschberg, A linear space algorithm for computing maximal common subsequences, Communications of the ACM 18 (6) (1975) 341–343.

[14] L. Wang, T. Jiang, On the complexity of multiple sequence alignment, Journal of computational biology 1 (4) (1994) 337–348.

[15] W. Just, Computational complexity of multiple sequence alignment with sp-score, Journal of computational biology 8 (6) (2001) 615–623.

[16] I. Elias, Settling the intractability of multiple alignment, Journal of Computational Biology 13 (7) (2006) 1323–1339.

[17] N. Saitou, M. Nei, The neighbor-joining method: a new method for reconstructing phylogenetic trees., Molecular Biology and Evolution 4 (4) (1987) 406–425. arXiv:https://academic.oup.com/mbe/article-pdf/4/4/406/11167444/7sait.pdf, doi:10.1093/oxfordjournals.molbev.a040454.
URL https://doi.org/10.1093/oxfordjournals.molbev.a040454

[18] D. G. Higgins, P. M. Sharp, Clustal: a package for performing multiple sequence alignment on a microcomputer, Gene 73 (1) (1988) 237–244. doi:https://doi.org/10.1016/0378-1119(88)90330-7.
URL https://www.sciencedirect.com/science/article/pii/0378111988903307

[19] M. A. Larkin, G. Blackshields, N. P. Brown, R. Chenna, P. A. McGettigan, H. McWilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez, et al., Clustal w and clustal x version 2.0, bioinformatics 23 (21) (2007) 2947–2948.

[20] R. C. Edgar, Muscle : multiple sequence alignment with high accuracy and high throughput, Nucleic acids research 32 (5) (2004) 1792–1797.
[21] D. J. Lipman, W. R. Pearson, Rapid and sensitive protein similarity searches, Science 227 (4693) (1985) 1435–1441.
[22] F. Cunningham, P. Achuthan, W. Akanni, J. Allen, M. R. Amode, I. M. Armean, R. Bennett, J. Bhai, K. Billis, S. Boddu, et al., Ensembl 2019, Nucleic acids research 47 (D1) (2019) D745–D751.

Annexe A. Résultats obtenus

Reconstruction complète																		
bait retrouvé			Plus de 21 résidus			Entre 14 et 20 résidus trouvés			Entre 7 et 13 résidus trouvés			Autres cas		
A	**!A**		**A**	**!A**		**A**	**!A**		**A**	**!A**		**A**	**!A**	
121540	**B**	**!B**	220	**B**	**!B**	959	**B**	**!B**	5864	**B**	**!B**	4376	**B**	**!B**
	618	30960		5	1053		26	5503		1243	79170		2323	84855
153118			1278			6488			86277			91554		

TABLE A.2 – **Résultats obtenus avec `alignBaitFusion` (en utilisant `ClustalW`).** La méthode `alignBaitFusion` renvoie toujours une séquence sans décalage (i.e. toutes les séquences reconstruites sont dites "complètes"). Ce tableau donne le nombre de **baits** retrouvés, le nombre de séquences avec plus de 21 résidus reconstitués (i.e. correspondants à ceux du *bait*), le nombre de séquences qui ont entre 14 et 20 résidus reconstitués et ainsi de suite. On distingue plusieurs cas. Le cas **A** est le cas où le *bait* est inclus dans les *baitModels* (voir section 4.1). Le cas **!A** est le cas où le **bait** n'est pas inclus dans les *baitModels*. Le cas **B** est le cas où au moins un *baitModel* est une séquence sans décalage de masse (mais différente du *bait* car $\mathbf{B} \subseteq \mathbf{!A}$). Le cas **!B** correspond à tous les autres cas.

Reconstruction complète																		
bait retrouvé			Plus de 21 résidus			Entre 14 et 20 résidus trouvés			Entre 7 et 13 résidus trouvés			Autres cas		
A	**!A**		**A**	**!A**		**A**	**!A**		**A**	**!A**		**A**	**!A**	
110572	**B**	**!B**	235	**B**	**!B**	955	**B**	**!B**	7368	**B**	**!B**	13829	**B**	**!B**
	100	3431		5	1062		23	4928		1019	61281		3068	130839
114103			1302			5906			69668			137736		

TABLE A.3 – **Résultats obtenus avec `alignBaitFusion` (en utilisant `MUSCLE`).** La méthode `alignBaitFusion` renvoie toujours une séquence sans décalage (i.e. toutes les séquences reconstruites sont dites "complètes"). Ce tableau donne le nombre de **baits** retrouvés, le nombre de séquences avec plus de 21 résidus reconstitués (i.e. correspondants à ceux du *bait*), le nombre de séquences qui ont entre 14 et 20 résidus reconstitués et ainsi de suite. On distingue plusieurs cas. Le cas **A** est le cas où le *bait* est inclus dans les *baitModels* (voir section 4.1). Le cas **B** est le cas où au moins un *baitModel* est une séquence sans décalage de masse (mais différente du *bait* car $\mathbf{B} \subseteq \mathbf{!A}$). Soit un cas x, le cas avec $!x$ est la négation du cas x.

Reconstruction complète																				
bait retrouvé				Plus de 21 résidus				Entre 14 et 20 résidus trouvés				Entre 7 et 13 résidus trouvés				Autres cas			
A	**!A**			**A**	**!A**			**A**	**!A**			**A**	**!A**			**A**	**!A**		
130379	**B**	**!B**		12	**B**	**!B**		62	**B**	**!B**		667	**B**	**!B**		1814	**B**	**!B**	
	1415	71527			5	403			31	1989			929	11116			1828	6822	
203321				420				2082				12712				10464			

Reconstruction partielle																			
Plus de 21 résidus				Entre 14 et 20 résidus trouvés				Entre 7 et 13 résidus trouvés				Autres cas							
A	**!A**			**A**	**!A**			**A**	**!A**			**A**	**!A**						
2	**B**	**!B**		1	**B**	**!B**		3	**B**	**!B**		19	**B**	**!B**					
	0	520			0	4563			0	56345			7	48256					
522				4564				56348				48282							

TABLE A.4 – **Résultats obtenus avec `linearBaitFusionV2`.** Les méthodes `linearBaitFusion` et `linearBaitFusionV2` renvoient soit une séquence sans décalage (i.e. reconstruction complète) soit une séquence avec un décalage de masse ou une séquence de résidus ambiguë (i.e. reconstruction partielle). Ce tableau donne le nombre de **baits** retrouvés, le nombre de séquences avec plus de 21 résidus reconstitués (i.e. correspondants à ceux du *bait*), le nombre de séquences qui ont entre 14 et 20 résidus reconstitués et ainsi de suite. On distingue plusieurs cas. Le cas **A** est le cas où le *bait* est inclus dans les *baitModels* (voir section 4.1). Le cas **B** est le cas où au moins un *baitModel* est une séquence sans décalage de masse (mais différente du *bait* car $\mathbf{B} \subseteq \mathbf{!A}$). Soit un cas x, le cas avec $!x$ est la négation du cas x.

Annexe B. Illustrations

Annexe B.1. SpecOMS

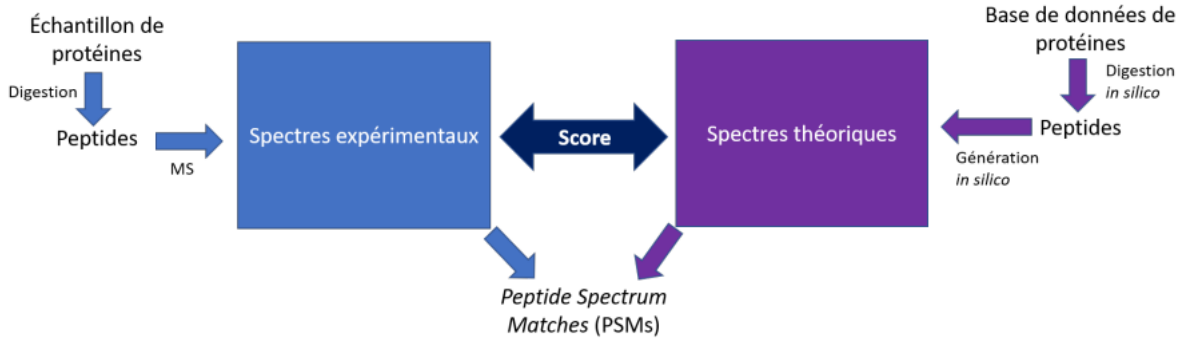

FIGURE B.1 – **Identification des spectres de masse par comparaison avec une base de données de spectres théoriques.** Cette identification comporte plusieurs étapes. Une fois que les spectres expérimentaux sont obtenus, ils ont confrontés à des spectres théoriques issus de la digestion d'une base de données de séquences de protéines. Pour cela, un score de comparaison est utilisé. À chaque spectre expérimental est affecté un ou plusieurs spectres théoriques qui sont sélectionnés par le score choisi (e.g. un seuil). Ces ensembles spectre-peptide sont nommés PSM. (*Schéma issu de* [5])

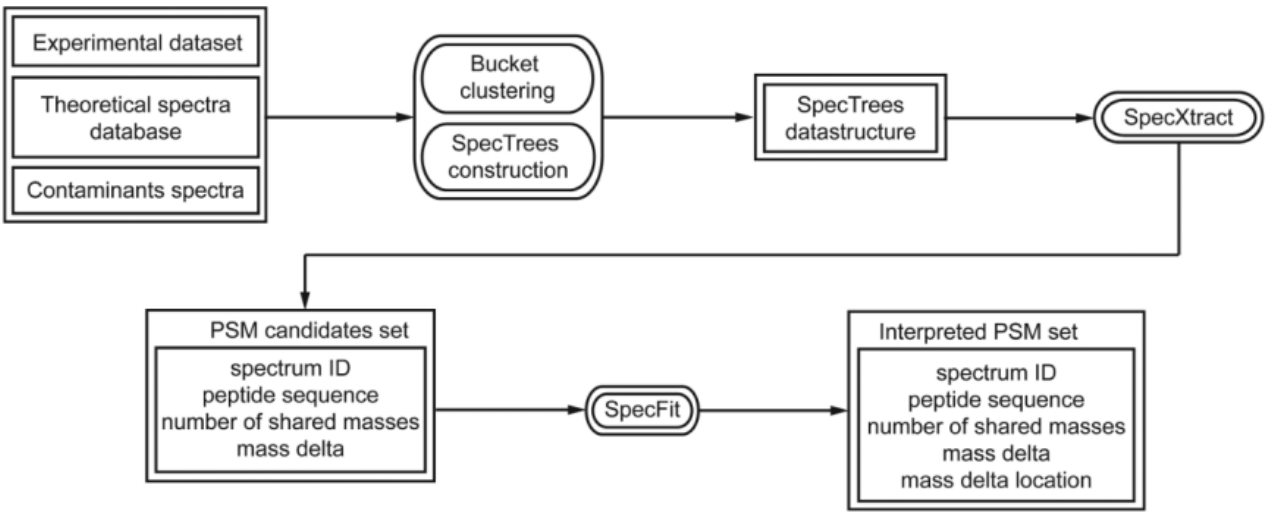

FIGURE B.2 – **Vue d'ensemble de SpecOMS.** Les rectangles représentent les données et les boîtes rondes représentent les processus. (*Schéma issu de* [6])

Annexe B.2. `linearBaitFusion`

Ci-après une illustration de la méthode `linearBaitFusion` sur le bait GGSGATIIMVVQR dont les baitModels sont GGSQTI[570.32]R, GGSGATII[457.24]R, NSG[44.07]VVMII[128.01]QR, GGSQTIIMVVQR et [114.04]SGATI[343.19]VQR. La position des curseurs est représentée par la couleur rouge. La séquence obtenue itérativement est représentée en bleu. Nous ne représenterons pas explicitement les états des baitModels ici mais nous préciserons lorsqu'ils

passent d'un état à un autre. Nous numéroterons les baitModels de 1 à 5 en partant du haut vers le bas (le baitModel 4 est donc le baitModel GGSQTIIMVVQR).

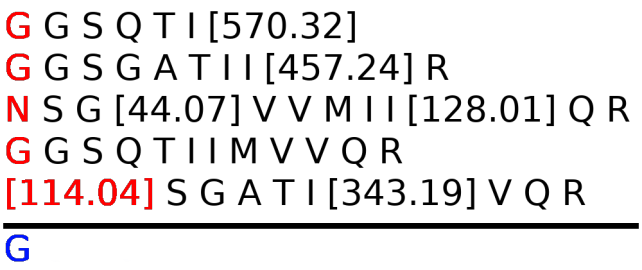

FIGURE B.3 – Initialement, tous les curseurs sont placés sur le premier élément de leur baitModel respectif. Les baitModels 1, 2 et 4 votent pour le résidu G. Le baitModel 3 vote pour le résidu N tandis que le baitModel 5 propose une masse de 114.04 Da. Le résidu G est élu à la majorité, la masse de G est donc soustraite à la masse proposée par le baitModel 5 et le baitModel 3 sera maintenant placé en probation parce qu'il propose un acide aminé différent de G. Les curseurs sont avancés d'un cran.

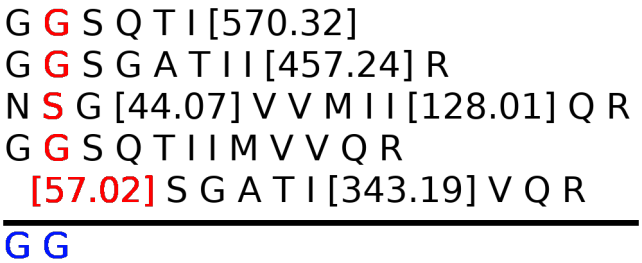

FIGURE B.4 – La masse proposée par le baitModel 5 correspond exactement à la masse du résidu G. Les baitModels 1, 2, 4 et 5 votent donc pour G. Le baitModel 3 précédemment en probation est maintenant invalidé car il propose à nouveau un résidu différent de l'élu. Un deuxième G est rajouté à la séquence de sortie. Les curseurs avancent d'un cran sauf celui du baitModel 3 qui est invalidé.

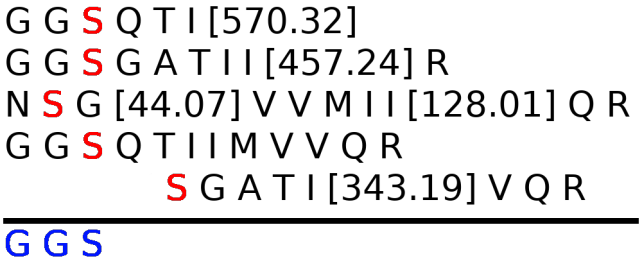

FIGURE B.5 – Les baitModels valides (1, 2, 4 et 5) proposent tous S. S est élu. Le curseur du baitModel 3, précédemment invalidé, est sur le résidu S donc à travers la stratégie "comeback" le baitModel 3 est replacé en probation. Les curseurs sont avancés.

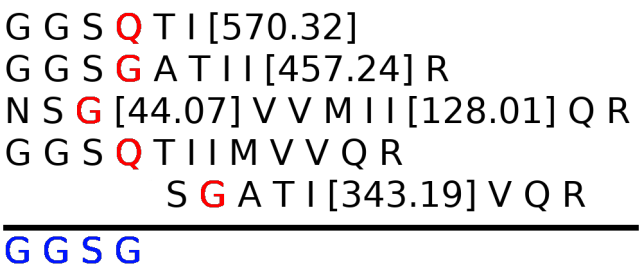

FIGURE B.6 – Les baitModels 1 et 4 votent pour le résidu Q, les baitModels 2, 3 et 5 votent pour G. G est élu. Les baitModels 1 et 4 sont placés en probation. Jusqu'ici les baitModels 1, 3 et 4 sont en probation tandis que 2 et 5 sont valides. Les curseurs sont avancés.

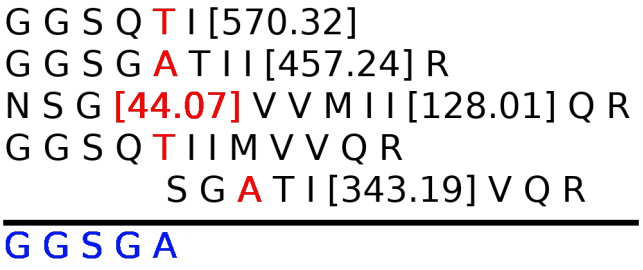

FIGURE B.7 – Les baitModels 1 et 4 proposent le résidu T, les baitModels 2 et 5 proposent le résidu A. T et A sont ex aequo lors de l'élection à la majorité. Le $score_A$ est utilisé pour les départager. Les baitModels 1 et 4 ($groupe_1$) sont en probation donc $score_A(groupe_1) = 2$ tandis que les baitModels 2 et 5 ($groupe_2$) sont valides d'où $score_A(groupe_2) = 4$. Le candidat du $groupe_2$ est alors élu, c'est-à-dire A. Les baitModels 1 et 4 sont alors éliminés parce qu'ils étaient en probation et sont à nouveau en désaccord. Le baitModel 3 propose une masse de 44.07 Da. Or, $44.07 - m(A) < -1$ donc la masse de 44.07 Da ne correspond à aucun résidu. Le baitModel 3 est définitivement invalidé. Les curseurs du baitModel 2 et 5 sont avancés.

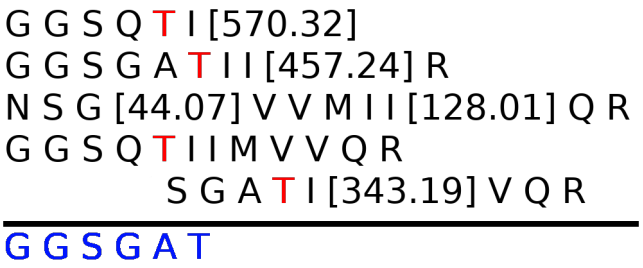

FIGURE B.8 – Les baitModels 2 et 5 proposent le candidat T. C'est le seul candidat donc il est élu. Les baitModels 1 et 4, précédemment invalidés, sont aussi sur l'acide aminé T donc il sont remis en probation. Tous les curseurs sont avancés sauf celui du baitModel 3 qui est maintenant détruit parce que le baitModel 3 est invalidé indéfiniment.

Nous présentons ci-après le résultat final après exécution de `linearBaitFusion`. Les étapes intermédiaires ne sont pas représentées ici. Ce travail est laissé au lecteur qui devrait retrouver le même résultat qu'à la FIGURE B.9.

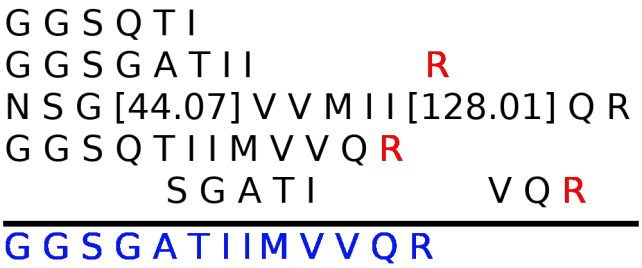

FIGURE B.9 – Quelques itérations après, les baitModels 2, 4 et 5 votent pour l'acide aminé R. R est élu. L'algorithme est terminé. La séquence reconstituée est GGSGATIIMVVQR et elle correspond exactement à la séquence du bait.