NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

INTERDISCIPLINARY POSTGRADUATE PROGRAM
"INFORMATION TECHNOLOGIES IN MEDICINE AND BIOLOGY"

MASTER THESIS

Algorithm for the construction of spliced-peptides database

Aliki Evangelia I. Konstantinou

Supervisors: **Dr. George Paliouras ,** Research Director at Institute of Informatics and Telecommunications, NCSR “Demokritos”

Dr. Stavros J. Perantonis, Research Director at Institute of Informatics and Telecommunications, NCSR “Demokritos”

ATHENS

FEBRUARY 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
"ΤΕΧΝΟΛΟΓΙΕΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΣΤΗΝ ΙΑΤΡΙΚΗ ΚΑΙ ΤΗ ΒΙΟΛΟΓΙΑ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλγόριθμος για την δημιουργία βάσης συναρμοσμένων πεπτιδίων

Αλίκη Ευαγγελία Ι. Κωνσταντίνου

Επιβλέποντες: **Δρ. Γιώργος Παλιούρας,** Διευθυντής Ερευνών, Ινστιτούτο Πληροφορικής και Τηλεπικοινωνιών, ΕΚΕΦΕ «Δημόκριτος»

Δρ. Σταύρος Περαντώνης, Διευθυντής Ερευνών, Ινστιτούτο Πληροφορικής και Τηλεπικοινωνιών, ΕΚΕΦΕ «Δημόκριτος»

ΑΘΗΝΑ

ΦΕΒΡΟΥΑΡΙΟΣ 2019

MASTER THESIS

Algorithm for the construction of spliced-peptides database

Aliki Evangelia I. Kostantinou

A.M.: 0154

SUPERVISORS **Dr. George Paliouras,** Research Director at Institute of Informatics and Telecommunications, NCSR "Demokritos"

Dr. Stavros J. Perantonis, Research Director at Institute of Informatics and Telecommunications, NCSR "Demokritos"

EXAMINATION COMMITTEE: **Dr. George Paliouras ,** Research Director at Institute of Informatics and Telecommunications, NCSR "Demokritos"

Dr. Stavros J. Perantonis, Research Director at Institute of Informatics and Telecommunications, NCSR "Demokritos"

Dr. Ema Anastasiadou, Lecturer-Researcher IV, Biomedical Research foundation of the Academy of Athens

Φεβρουάριος 2019

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλγόριθμος για την δημιουργία βάσης συναρμοσμένων πεπτιδίων

Αλίκη Ευαγγελία Ι. Κωνσταντίνου

Α.Μ.: 0154

ΕΠΙΒΛΕΠΟΝΤΕΣ **Δρ. Γιώργος Παλιούρας,** Διευθυντής Ερευνών, Ινστιτούτο Πληροφορικής και Τηλεπικοινωνιών, ΕΚΕΦΕ «Δημόκριτος»

Δρ. Σταύρος Περαντώνης, Διευθυντής Ερευνών, Ινστιτούτο Πληροφορικής και Τηλεπικοινωνιών, ΕΚΕΦΕ «Δημόκριτος»

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ **Δρ. Γιώργος Παλιούρας,** Διευθυντής Ερευνών, Ινστιτούτο Πληροφορικής και Τηλεπικοινωνιών, ΕΚΕΦΕ «Δημόκριτος»

Δρ. Σταύρος Περαντώνης, Διευθυντής Ερευνών, Ινστιτούτο Πληροφορικής και Τηλεπικοινωνιών, ΕΚΕΦΕ «Δημόκριτος»

Δρ. Έμα Αναστασιάδου, Ερευνήτρια Δ', Ιδρυμα Ιατρονιολογικών Ερευνών Ακαδημίας Αθηνών (ΙΙΒΕΑΑ)

February 2019

ABSTRACT

The identification of spliced peptides has garnered significant interest in the recent years due to their role in the function of adaptive immune system, especially in the context of cancer. Spliced peptides are composed from protein fragments originally distant in the parental protein. Splicing takes place in the proteasome, a protein complex in the cell that regulates the concentration of unneeded, damaged or pathogenic proteins. Proteolysis results in the degradation of proteins to smaller peptides of 7-12 amino acids that are either further degraded into amino-acids for the composition of new proteins or transferred onto the cell surface to be recognized by the immune system. Those peptides where believed to be only linear fragments of the parental proteins, however recent studies proved otherwise. They can derive from distant peptidic fragments of a protein sequence.

The experimental identification of spliced peptides requires Mass Spectrometry (MS) analysis and computational software that matches the spectra to theoretical peptides. The latter process requires a proteome or peptide database containing the sequences assumed to be present in the experiment. However, there are no readily available computational solutions to construct a complete spliced-peptides database. Therefore, the database construction should be implemented by the user who conducts a related experiment.

The current thesis aims to overcome this computational gap. We developed a methodology, implemented as an R package, that enables the user to construct a custom spliced-peptide database, save it in a convenient file format according to MS analysis software requirements and reduce its size to occupy less disc space and speed up the analysis process. Additionally, the database computing can be parallelized for optimized speed. Lastly, identified peptides from spectra matching can be searched against the database to verify their origin and estimate the false discovery rate (FDR).

SUBJECT AREA: Computational Proteomics

KEYWORDS: spliced peptides, immune system, autoimmune diseases, peptide database, R package

ΠΕΡΙΛΗΨΗ

Η αναγνώριση των συναρμοσμένων πεπτιδίων (spliced peptides) έχει κερδίσει έντονο ερευνητικό ενδιαφέρον τα τελευταία χρόνια λόγω του καταλυτικού τους ρόλου στην λειτουργία του ανοσοποιητικού συστήματος και σε μία πληθώρα αυτοάνοσων νοσημάτων, ειδικότερα του καρκίνου. Τα συναρμοσμένα πεπτίδια προκύπτουν από την ένωση απομακρυσμένων πρωτεϊνικών τμημάτων στο πρωτεάσωμα, το οποίο αναλαμβάνει την απορύθμιση άχρηστων, επιβλαβών ή κατεστραμμένων πρωτεϊνών στο κύτταρο. Μέσω της πρωτεόλυσης επιτυγχάνεται η διάσπαση των πρωτεϊνικών μορίων σε μικρότερες πεπτιδικές αλυσίδες 7-12 αμινοξέων οι οποίες είτε οδηγούν σε ανακύκλωση των αμινοξέων για σύνθεση νέων πρωτεϊνών είτε μεταφέρονται στην επιφάνεια του κυττάρου για να αναγνωριστούν από το ανοσοποιητικό σύστημα. Τα πεπτίδια αυτά, πρόσφατα αποδείχθηκε ότι δεν είναι αποκλειστικά συνεχόμενες αμινοξικές αλυσίδες της γονικής πρωτεΐνης αλλά μπορούν να προκύψουν από απομακρυσμένα μεταξύ τους πεπτιδικά τμήματα.

Η πειραματική διαδικασία της αναγνώρισής των συρραμμένων πεπτιδίων απαιτεί την χρήση φασματομετρίας μάζας (Mass Spectrometry, MS) και μία σειρά από υπολογιστικά εργαλεία τα οποία στοχεύουν στην ταυτοποίηση των φασμάτων. Για τις υπολογιστικές διαδικασίες απαιτείται η χρήση μίας βάσης δεδομένων που περιλαμβάνει όλες τις πρωτεϊνικές ακολουθίες που αναμένεται να είναι παρούσες στο πείραμα. Τα υπάρχοντα υπολογιστικά εργαλεία μέχρι σήμερα, δεν προβλέπουν την δημιουργία μιας ολοκληρωμένης βάσης συναρμοσμένων πεπτιδίων για μία δεδομένη βάση πρωτεϊνών. Συνεπώς η διαδικασία της δημιουργίας της μεταφέρεται στον χρήστη που θέλει να χρησιμοποιήσει τα συγκεκριμένα εργαλεία για την ανάλυση του πειράματός του.

Στην παρούσα εργασία, αναπτύχθηκε μία μεθοδολογία η οποία επιτρέπει στον χρήστη να δημιουργήσει μία βάση συναρμοσμένων πεπτιδίων προσαρμοσμένη στις ανάγκες του πειράματος. Επιπλέον δίνει την επιλογή της προσαρμοσμένης μορφοποίησης της βάσης ανάλογα με τις απαιτήσεις του λογισμικού που χρησιμοποιείται για την ταυτοποίηση των πεπτιδίων, τη δυνατότητα του περιορισμού του μεγέθους της και την παραλληλοποίηση της διαδικασίας για μέγιστη ταχύτητα εκτέλεσης. Τέλος δίνει τη δυνατότητα στο χρήστη να αναζητήσει τα ταυτοποιημένα πεπτίδια πίσω στη βάση ώστε να εξακριβωθεί η προέλευσή τους αλλά και να εκτιμηθεί το ποσοστό των ψευδώς θετικών αποτελεσμάτων. Ο αλγόριθμος έχει υλοποιηθεί σε μορφή R πακέτου.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Υπολογιστική Βιολογία, Πρωτεομική

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: συναρμοσμένα πεπτίδια, ανοσοποιητικό σύστημα, αυτοάνοσα νοσήματα, βάση πεπτιδίων, πακέτο R

CONTENTS

LIST OF FIGURES

INTRODUCTION

1.1 Thesis purpose

The current thesis was an effort of overcoming a methodological gap that exists in computational proteomics, regarding the experimental identification of spliced peptides. The splicing process of proteins in the proteasome is a relatively new but novel finding. This concept being so new might be a reason why computational solutions for identification are not yet readily available or complete.

Proteolysis, which is the breakdown of proteins inside the cell and takes place inside the proteasome, results in peptidic sequences and serves several purposes. It might take place in post-translational process to activate a protein, to break down proteins from food to provide amino acids in the organism or to regulate the accumulation of unneeded proteins inside the cell. Here we focus on its critical role in the function of the adaptive immune system.

A fraction of the peptides derived from the breakdown of proteins is transferred onto the cell surface to act as flags for the immune system. When peptides derived from viral, tumoral or pathogenic proteins and processes are recognized, the cells displaying those peptides are destroyed. Till very recently, epitopes found on the cell surface had been believed to be only continuous fragments of proteins. However this changed when several peptides composed of distant fragments, called spliced-peptides, were identified.

The process of identifying peptides in an experiment includes Mass Spectrometry analysis which results in spectra of masses and probabilistic matching of those spectra to actual peptidic sequences. The latter step requires a theoretical database of the proteins or peptides assumed to be present in the experiment.

Computational software available for proteomics is not yet configured to compute the vast databases associated with spliced-peptides. Here we developed ProteoSplicer, an R package that enables the user to construct and further process a custom spliced-peptide database and finish the downstream analysis by identifying the matched peptides back in the database, while requiring minimum knowledge of programming. It is developed in accordance with Liepe's described workflow [8]. It is compatible with any type of system configuration and operating system and developed to execute in a reasonable amount of time. The processes are parallelized to use as many cpu cores the user wants to dedicate.

The following computational tasks can be performed:

- Construction of custom spliced peptide database for varying peptides lengths and intervening sequences lengths.

- Reduction of the database by filtering out peptides that are not potential candidates according to their mass, with user defined tolerance of filtering.

- Construction of decoy databases using reliable methods known in the current literature.
- Identification of matched peptides in the database.

1.2 Thesis structure

In Chapter 2. the background of the subject is explained. We describe the concept of peptide splicing and its importance in the current knowledge of immune system function. We provide all the literature available that has focused on the identification of spliced peptides and its methodology. We also mention in an introductory manner the current computational tools that exist for this purpose.

In Chapter 3 we explain in depth every technical aspect of the algorithm. We provide a guide on how the user can run the algorithm step by step, how each function works and what outputs are computed.

In Chapter 4 we benchmark all the heavily computational tasks and provide an accurate estimation of execution time, depending on the parameters and the length of database.

2. BACKGROUND

2.1 Peptides and the immune system

The proteasome is a protein complex responsible for degradation of damaged or unneeded proteins inside the cell. Those proteins are tagged with another small protein, ubiquitin, by ubiquitin ligases. This signals the degradation process which results in small peptides of around 7-12 amino-acids.

A fraction of those peptides is transferred into the lumen of the Endoplasmatic Reticulum (ER) by a transporter associated with antigen processing (TAP) [1]. There, they are loaded onto major histocompatibility complex (MHC) class 1 molecules (also called human leukocyte antigen, HLA) which then exit the ER and reach the cell surface. As a result, peptides originating from tumoral or viral proteins are displayed on the surface of the cell by MHC and, as the immune system constantly monitors cellular integrity, CD8 cytolytic T lymphocytes (CTL) recognize those peptides and destroy the cell marked by them [2].

Thus, the presence of epitopes on the cell surface of eukaryotic organisms is the major mechanism through which pathogens, tumoral activity and any kind of malfunctioning is identified.

2.1.2 Peptide splicing in the proteasome

Antigenic peptides recognized by CTL, has been believed to originate only from linear fragments of the parental proteins. However, peptides composed of distant protein fragments assembled together by a new peptide bond, have been recently identified although considered to be a rare event. This process of peptide splicing has been shown to take place in the proteasome. Most of them fall within the usual length range of 8-12 amino-acids, as the average length of proteasome generated peptides is 11 amino acids, however spliced peptides of a longer amino-acid chain have been identified as well.

The first spliced peptide was discovered by Hanada K. et al .who showed that a CTL clone recognized a peptide derived from FGF-5 protein and generated by protein splicing, when it was targeted towards a renal cell carcinoma [3]. Despite its unusual length (40 amino-acids) it was shown later to be produced by the proteasome [4]. Vigneron et al. also identified a second peptide composed by two non-linear fragments of the melanoma differentiation antigen gp100 [4] and then a third peptide containing two fragments of the SP110 protein, spliced together in the reverse order than the original sequence [5]. All splicing reactions have been shown to take place in the proteasome by transpeptidation and the intervening sequence between the fragments ranged from 4 to 40 amino acids. Vigneron et al. tested the efficiency of the splicing over different lengths of the intervening sequence and observed that the longer the length, the lower the efficiency of the splicing process [6]. Moreover, they showed that splicing with fragments from different parental proteins hardly ever occurred as it is highly unlikely that two distinct protein substrates can be degraded simultaneously in the same proteasome[6]. The same group, recently identified a fifth antigenic spliced peptide originated from protein gp100 [7].

2.1.3 A Computational approach

In principle, a spliced peptide could occur via transpeptidation between any two protein fragments released through proteolysis. As expected, the list of all the possible splicing combinations for a given proteome database can become vast. However, until recently, only few spliced peptides have been in vitro identified. This could be the result of two limitations: the availability of specific CTL directed against spliced peptides and the lack of a spliced peptide database. Mass spectrometry software used for matching ions is limited to produce only linear protein fragments.

Liepe et al. developed an analytical strategy that could handle the vast spliced-peptide database derived from the human proteome and analyzed the HLA cass 1 immunopeptidome of three unrelated cell lines. They concluded to the observation that around one quarter of the total number of the peptides found at the cell surface, was represented by spliced peptides. They account for one-fourth in terms of abundance and one-third in terms of diversity [8]. Espe cially in the case of melanoma-associated epitopes, they reported that the abundance of spliced peptides was comparable to that of the non-spliced.

In previous studies, it was suggested that the generation of spliced-peptides might rely on specific underlying mechanisms. One of them might be the preference for specific peptide sequences [4] [5] [9] [10]. However the limited findings could not draw any conclusions with sufficient statistical power. This large pool of spliced peptides, obtained in Liepe et al. study, resolved this issue. It was observed that there were no differences in the frequencies of the parental proteins of all the antigenic epitopes, no significant preference for specific antigens generating non-spliced versus spliced peptides, no difference in peptide length distribution between spliced and non-spliced and, finally, no preference for specific intervening sequence length within the fragments of spliced peptides. The peptides' motifs were also studied to determine whether there is a specific binding affinity to HLA-I molecules. Spliced peptides found to be non-compatible with HLA-A and HLA-B variants which was also supported in previous peptide-splicing analysis [11].

In addition, it was found that several proteins were represented only through spliced peptides rather than linear epitopes suggesting that, those proteins, may fail to produce linear fragments with binding motifs to HLA molecules. Antigen targeting only through non-spliced peptides may be limited due to sequence restrictions, thus, peptide splicing mechanism might be an evolutionary mechanism to combat this.

Liepe et al. managed to circumvent the previously mentioned computational limitations by creating a custom, theoretical, spliced peptides database [8] .To reduce the size of this database:

(i) the intervening sequence between two fragments was at most 25 amino-acids;

(ii) the range of peptide length was 9-12 amino acids as those constitute the large majority of HLA-I immunopeptidome;

(iii) post translational modifications of the peptides were ignored;

(iv) spliced peptides derived from fragments of two distinct proteins were excluded;

(v) theoretical peptides with computed molecular weights not equal to the masses detected in the experiment, where filtered out of the database.

The workflow of the identification of all peptides eluted from the HLA-1 molecules consisted of the following steps, as described in the supplementary file:

1. Extraction of all the spliced and non-spliced, 9 to 12 amino-acids, peptides from the human proteome database. The database of the spliced peptides included all possible protein-fragments combination, in both normal (N-terminal splice-reactant to C-terminal splice reactant) and reverse order, for intervening sequence lengths of 1 to 25 amino-acids.

2. Computation of the molecular weights for all entries in both non-spliced and spliced databases.

3. Extraction of the precursor masses from the Mass Spectrometry analysis, in order to obtain an experimental mass list.

4. The entries in non-spliced and spliced databases obtained from step 1. that had a molecular weight equal to at least one entry in the experimental mass list were kept in the database (using 3 ppm tolerance).

5. Transformation of both databases into FASTA format, with a structure that followed human proteome. That was several peptides each time bound together in a sequence of adequate length to form a protein block.

6. Construction of a decoy for each database by randomizing their protein sequences.

7. All databases (spliced, non-spliced, decoys) merged into one. This was used as the input in a MS search engine (Mascot) along with the raw data from the MS experiment. In the search engine, the protein entries provided, were cut into peptides again, either randomly or by rule, and matched via probabilistic rules to the ions provided in the experimental data, providing a list of peptides that were the best candidates.

8. The candidate matched peptides were then searched against the non-spliced, spliced and decoy databases with the peptide entries (obtained from step 1.), in order to determine if they were actual peptides and not decoys or sequences containing two separate peptide fragments as a result of the stitching step.

2.2 Computational tools in Proteomics

Proteomics is the study of all proteins in a biological system. Usually the interest is held for specific biological events or conditions, for example the protein expression, in variety and abundance, during cell mitosis or in tumor cells. To identify and quantify protein and peptide molecules, the most common analytical technique used is a combination of tandem Mass Spectrometry (MS/MS) and sequence database searching.

A mass spectrometer ionizes the chemical species provided in the sample and sorts the ions based on their mass-to-charge (m/z) ratio. In the case of a protein sample, the molecules undergo a random fragmentation resulting in smaller and charged amino-acid chains. Each one of them provides a mass spectrum, a plot of the ion signal as a function to the m/z ratio. In order to match the spectra obtained to an amino-acid sequence and, therefore, to the best protein candidate, specialized MS analysis software is needed.

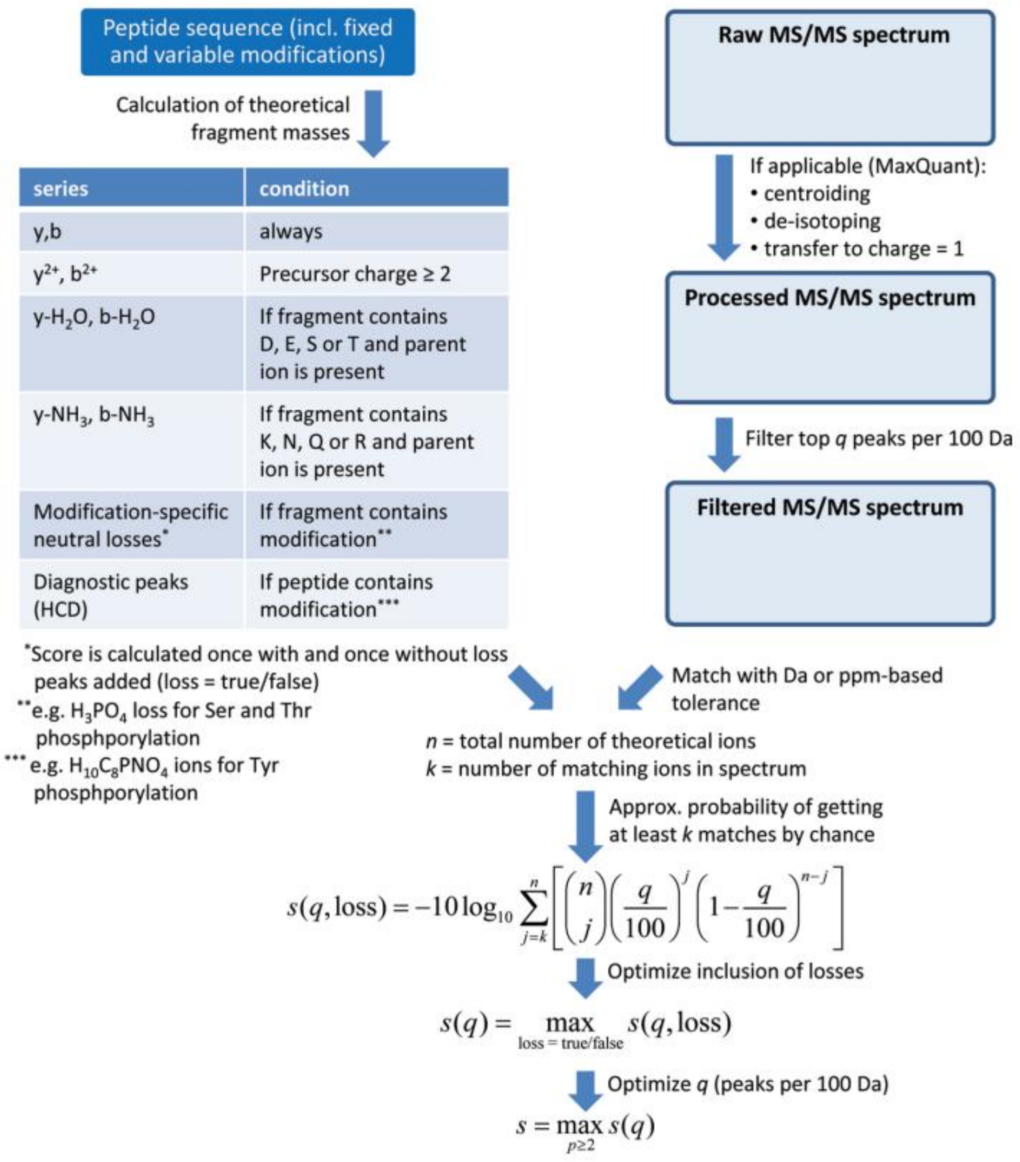
The peptide identification algorithms in MS software fall into two classes: de novo search and database search. The latter searches the spectra against a database that contains all the amino-acid sequences assumed to be present in the experimental sample. For each spectrum, it assembles a list of matched known peptide sequences through a scoring function. The peptide with highest score is considered the best match [12]. This is the most popular approach and numerous related software packages are available. Mascot, MaxQuant, Sequest , ProLuCID are a few popular examples.

De novo search is usually based on graph theory, as it was first described by Bartels [13]. In this method peaks in the spectrum are transformed into graph vertices. If two vertices in this spectrum graph have the same mass difference of one or several amino acids, a directed edge is applied. Other methods have also been proposed, including:

i) Composition of list with all possible peptides for a given spectrum and then matching each candidate's theoretical spectrum to the experimental. The most similar spectrum is the most likely to be the right sequence [14][15]
ii) Subsequencing. A method which matches short sequences of peptides that represent only a part of the complete peptide. Sequences that match the fragment ions in the experimental spectrum are extended one by one amino-acid until the best match is found [16] [17].
iii) A method which displays all series of peaks differing by the mass of each amino-acid residue. Fragment ions that have the same mass differences of one amino acid are connected by lines. This can be helpful for manual de novo peptide sequencing. [18]

In database search software each set of tandem mass (MS/MS) spectra, acquired from the mass spectrometer, is interrogated against a theoretical mass spectrum derived from peptides in the database. For each observed spectrum, candidate peptides are retrieved from the input database according to their theoretical mass so that it matches the precursor mass of the observed spectrum. As expected, if a database is large, many candidate peptides will be identified. The pairing of a single input spectrum with a single candidate peptide is termed a *peptide-spectrum match* (PSM) and requires a scoring function which computes a specific score for each theoretical spectrum. Each software solution uses its unique scoring algorithm which should usually be a function of the total number of theoretical ions and the number of matching ions in the spectrum.

In **Figure 2.1** is the scoring algorithm representation of Andromeda search engine in Mascot [19].


Peptide sequence (incl. fixed and variable modifications)
Calculation of theoretical fragment masses
Raw MS/MS spectrum
If applicable (MaxQuant):
• centroiding
• de-isotoping
• transfer to charge = 1
Processed MS/MS spectrum
Filter top *q* peaks per 100 Da
Filtered MS/MS spectrum
Match with Da or ppm-based tolerance
Approx. probability of getting at least *k* matches by chance
Optimize inclusion of losses
Optimize *q* (peaks per 100 Da)

series	condition
y,b	always
y^{2+}, b^{2+}	Precursor charge ≥ 2
y-H_2O, b-H_2O	If fragment contains D, E, S or T and parent ion is present
y-NH_3, b-NH_3	If fragment contains K, N, Q or R and parent ion is present
Modification-specific neutral losses*	If fragment contains modification**
Diagnostic peaks (HCD)	If peptide contains modification***

*Score is calculated once with and once without loss peaks added (loss = true/false)
**e.g. H_3PO_4 loss for Ser and Thr phosphporylation
*** e.g. $H_{10}C_8PNO_4$ ions for Tyr phosphporylation

n = total number of theoretical ions
k = number of matching ions in spectrum

$$s(q,\text{loss}) = -10\log_{10}\sum_{j=k}^{n}\left[\binom{n}{j}\left(\frac{q}{100}\right)^{j}\left(1-\frac{q}{100}\right)^{n-j}\right]$$

$$s(q) = \max_{\text{loss = true/false}} s(q,\text{loss})$$

$$s = \max_{p\geq 2} s(q)$$

Figure 2.1: Andromeda scoring algorithm [19]

2.3 Target-Decoy search strategy

As described previously, the output of mass spectrometry analysis software indicates the most likely theoretical peptide matches to the input spectra. Those are then used to infer the parental protein sequence that was present in the biological sample. However there are several limitations interfering with the accuracy of the results; not all peptide species in a sample are represented in the search database; non-peptide species in the input spectra will be falsely given a peptide assignment and lastly, an incorrect candidate might, occasionally, be given the highest score in the candidate list. It is crucial that an extra step in the downstream analysis should be taken in order to distinguish the false positives and estimate their frequencies. Target decoy methods are powerful, yet simple tools for this purpose.

There are five criteria a successful decoy database should meet [20].

1. There should be similar amino acid distributions to the target protein sequence.
2. Similar protein length distribution to the input protein database.
3. Similar numbers of proteins as target protein list.
4. Similar numbers of predicted peptides.
5. No peptides in common.

There are several methods to construct a decoy:

1. *Reversed sequences*. The simplest and most widely used method that produces the reversed sequences [21][22]. For example the decoy sequence for "ABCDEFGHIJKL" would be "LKJIHGFEDCBA". As it switches the amino-carboxyl orientation of the protein's amino-acids, the number of actual peptide sequences preserved is virtually zero. It is fairly simple to programmatically implement so it is easily replicable between different researchers. However, it is not a random database and does not represent a null random distribution which might be needed for certain peptide types. Nonetheless, it can easily be deduced that it meets all five criteria mentioned above [23].

2. *Shuffled Proteins.* Another reliable method which also has the stochastic properties lacking in the previous one [24]. In this method the amino-acids of a protein sequence are sampled and randomly rearranged. For example, in sequence "ABCDEFG" a possible decoy would be "CAGFDBE". It is also very easily implemented programmatically and it preserves all the parental protein features meeting all the criteria described above. However redundancies and homologies between protein entries will not be preserved, resulting in a greater number of decoy peptides than originally present in the target sequence list [20].

3. *Random Proteins.* Sequences generated in a completely random manner. Amino acids and length sampling should follow the distribution that is normally found in actual proteins. Thus, for a given protein database, the frequency matrix of amino-acids and the distribution of proteins' lengths should be first estimated. For example

A random sequence using alphabet letters would be “AJTKROELHTEBBPEFWD”. However each letter has a specific probability to be chosen during sampling, according to the frequencies found in the original input sequence.

4. *Decoy peptides.* In all three previous methods, the decoy database consists of decoy proteins which then are “digested” into peptides by the MS analysis software used. An alternative way is to generate decoy peptides by altering directly the peptides derived from the protein list (reversing or sampling) instead of the parental sequences. This way, the decoy peptides’ masses are the exact same as the peptides that the search engine will consider for matching.

2.3.1 Separate versus composite database.

There are two database search strategies when estimating the false positive rate once a decoy is constructed. Separate and combined search.

One method is to supply the search engine with two separate databases: the target and the decoy. This way, each input spectrum is searched against two databases and has one target and one decoy best score. However this method can lead to an overly conservative interpretation of search results. Without competition between decoy and target sequences for the top-ranked score, it is likely that decoy sequences that partially match MS/MS spectra, will get a higher score compared to target-search hits. Also, all peptides that are below the score at which decoy hits outnumber target hits, are assumed to be incorrect. This way the false positive rate is overestimated [23].

It is generally accepted to construct one single database that consists of both target and decoy sequences and clearly annotating each sequence’s origin. In this approach, both target and decoy peptides compete each other for the best score and incorrect peptide matches will be randomly drawn from decoy and target sequences.

3. METHODOLOGY

In this chapter, we explain the method that constructs the databases needed in the process of spliced-peptides identification and then describe step by step the implementation of the procedure. The databases obtained are useful when a database search algorithm (as described in chapter 2) is used to match experimental mass spectra to theoretical peptides. The pipeline follows the workflow of Liepe et al. [8], presented in the section 2.1.3.

3.1 The Method

In the spliced-peptide identification process, one major issue is the unavailability of a complete database, as its size can become quite large depending on the experiment and the peptide lengths needed to be studied. Therefore, it is mandatory that for any related experiment, the researcher constructs its own custom database. The following method has been developed to provide a custom spliced and non-spliced database, according to user parameters. The procedure consists of the following steps that are described in detail in section 3.2 :

1. Construction of both spliced and non-spliced databases for a given proteome database. For each protein sequence, first all linear single fragments of the desired length are derived to obtain the non-spliced peptides. Then, all possible paired AA fragments combinations are computed according to a minimum and maximum intervening sequence to obtain the spliced-peptides database. If non-spliced peptides also appear in the spliced database they are removed from the latter.
2. Reduction of the obtained databases by filtering out the peptides whose molecular weights are not represented in the experimental mass list obtained from MS/MS analysis. This step might result in a significant reduction depending on the tolerance selected for filtering.
3. Construction of spliced and non-spliced decoy databases.
4. Saving all databases in two formats: i) lists of peptides along with their AA indices to their parental protein and ii) FASTA files of the peptides stitched together to resemble proteins.
5. This is the last and separate step in the identification analysis: after using MS analysis/ database search software, the peptides that have been found to match the MS/MS spectra are searched against the spliced, non-spliced and decoy databases saved in step 4, in order to determine their origin.

3.2 Implementation of the procedure

As the process has high computational requirements, it is parallelized, in order to exploit the full capabilities of available hardware and optimize speed. However, the option of running in serial mode is also available. Here we explain the method used and the challenges faced during the development.

3.2.1 Automated process

There are two options for running the workflow. Either one function is called that automatically performs all the tasks according to user's settings requiring minimal user involvement, or each function is called separately and the user can have control of each task. The latter is useful when it would be best to create once a database that can be later used for different experiments.

To initiate the automated process the following function is called:

```
>create_database ( data, nmers, isl,
                   target, filter, masslist,
                   tolerance, as.blocks, decoy,
                   ncores )
```

The parameters are the following:

1. ***data:*** the input file in FASTA format which contains all the proteins assumed to be present in the experiment.

2. ***nmers(min, max)*** : a numeric pair including the minimum and maximum length of peptides. If peptides of a specific length should be produced, this length should be defined as the minimum and maximum, i.e. c(9,9) for peptides of 9 amino-acids only.

3. ***Isl (min, max)*** : intervening sequence length. Also a pair of the minimum and maximum of the sequence length that may intervene between two peptide fragments.

4. ***target*** : a string variable with three possible values: "spliced" if only the spliced database is needed; "non-spliced" if only the non-spliced database is needed; "all" if both databases should be saved.

5. ***filter*** : Boolean variable. If TRUE the database is reduced by filtering the entries according to their molecular weight and the experimental mass list provided by the user.

6. ***masslist*** (optional): a numeric vector containing all the ion masses found in the MS analysis. These can be obtained either from conversion of the raw spectra files or by MS analysis software that matches the masses to the proteome.

7. ***tolerance***(optional): numeric variable defining the tolerance in p.p.m. used to filter the peptides according to mass list provided.

8. ***as.blocks***: Boolean variable. If TRUE the dataset is also saved in FASTA format with a proteome structure (peptides stitched together to resemble proteins)

9. ***decoy***: Boolean variable. If TRUE a decoy database is also created using reverse or shuffled sequences.

10. ***ncores***: Numeric variable defining the number of cores in the system or cluster to be used in parallel processing. When $ncores = 1$ the process will be serial.

The function first estimates the proteome length distribution and its parameters (mean and variance) which are used later on to sample protein lengths for generating peptide-proteins (only when parameter *as.blocks* == *TRUE*). The distribution estimation excludes outliers, i.e. unusually large proteins that contain more than 20x the number of amino-acids than the median length.

Afterwards, the proteome is assorted in groups of proteins of varying lengths that follow the original proteome distribution, so that every group accounts for roughly the same workload. This is done by sorting ascending the proteins according to length and filling each group with protein $p = i$ to protein $p = n$ while in-between proteins are separated from one another by $k = K$ proteins in the rank. For example, if we choose to have 20 groups of proteins in a proteome of 20000 proteins then each group should contain 1000 proteins, one protein every 20 in the sorted list. The first group gets the proteins $p = 1,\ p = 21,\ p = 41,\ p = 61, \ldots.,\ p = 19981$, the second group $p = 2, p = 22, p = 42, p = 62, \ldots, p = 19982$ etc. This way all groups follow the length distribution of the proteome as illustrated in **figure 3.1**

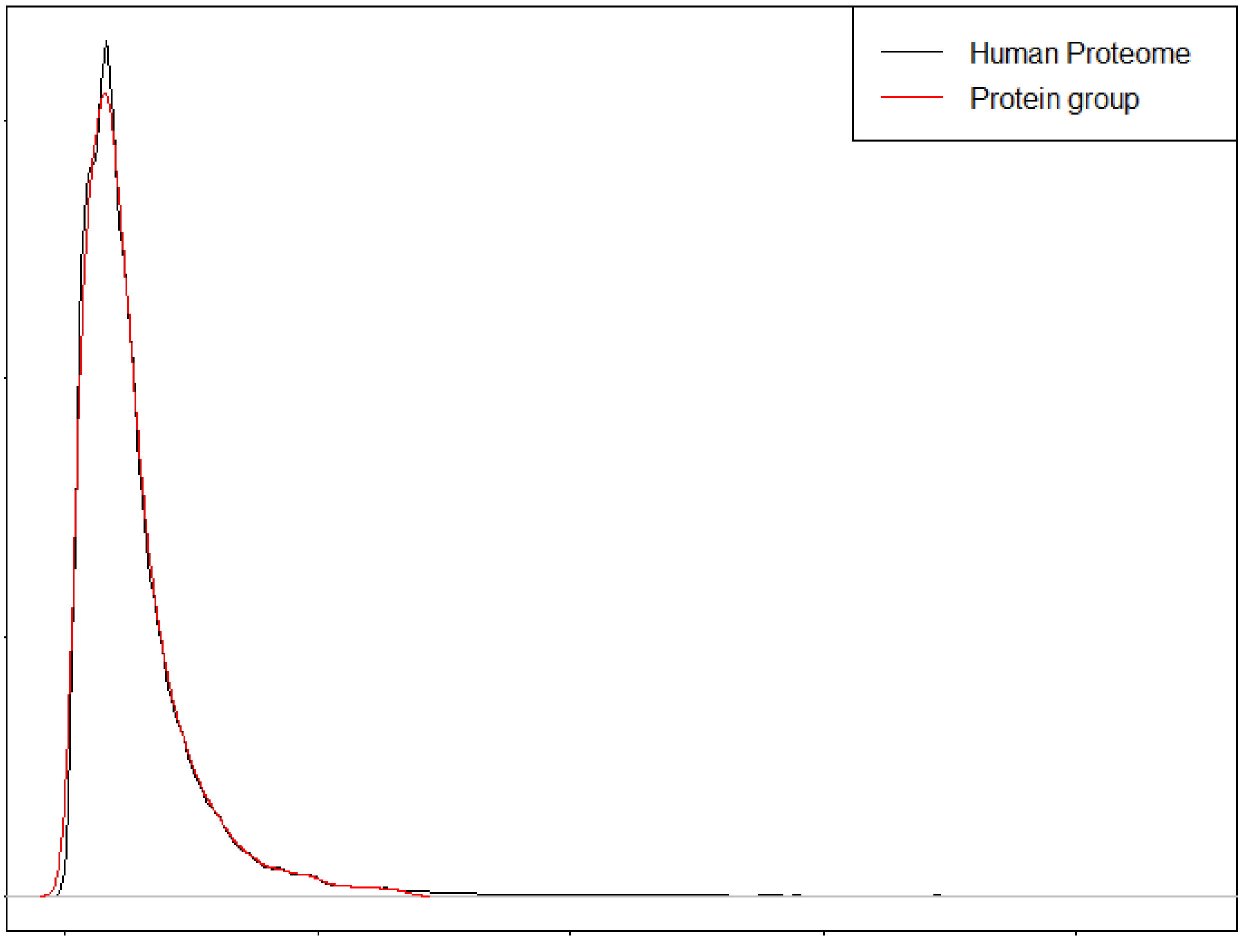


Figure 3.1 Length distribution of proteome versus group of proteins

This facilitates the monitoring and the runtime estimation of the process as all groups require equal computing time. Even though the algorithm has been optimized for speed, monitoring is especially useful when a single machine is used along with other processes and the database computing might need to be paused. The user can keep track of the groups that have already been processed and restart from the last processed protein group. The status of the process is printed in a .txt file in the same directory.

In the last step, the function creates all folders and subfolders where the databases will be saved and gives initiation to the parallel computing.

The interdependencies of functions are illustrated in **figure 3.2**

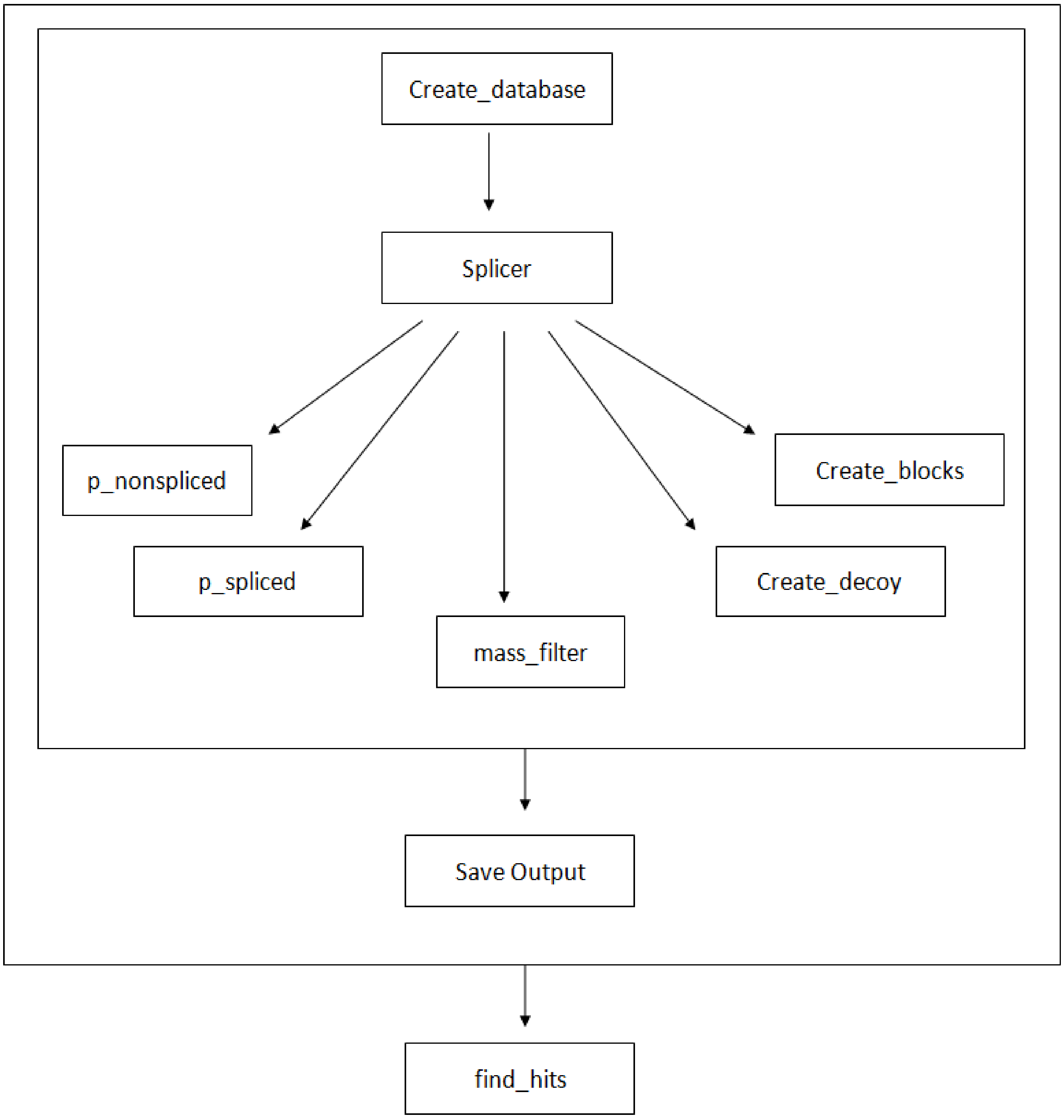


Figure 3.2: Illustration of workflow and interdependencies of functions

The majority of the computing work takes place within the *splicer()* which in turn calls all the functions needed for each individual task i.e. peptide database computation, filtering according to mass list, decoy construction and output management for a given protein. In *splicer()* all the steps described below (3.1.4 to 3.1.7) are automatically completed, deriving in reduced, filtered and ready to use spliced, non-spliced and decoy databases for a given protein. It is designed to be called within a serial or parallel process and includes the task of monitoring.

In the next sections we explain how each function works, what it requires as input and what it returns as output.

3.2.2 Protein catalyzing and splicing.

The following functions are called for the peptides' computation:

>*p_nonspliced(sequence, N)* and

>*p_spliced(sequence, N, isl)*

Where:

- ***sequence*** is the input sequence in a vector of characters ["A", "B", "C"…],
- ***N*** is the number of amino-acids for each peptide,
- ***isl*** a vector of the minimum and maximum length intervening sequence.

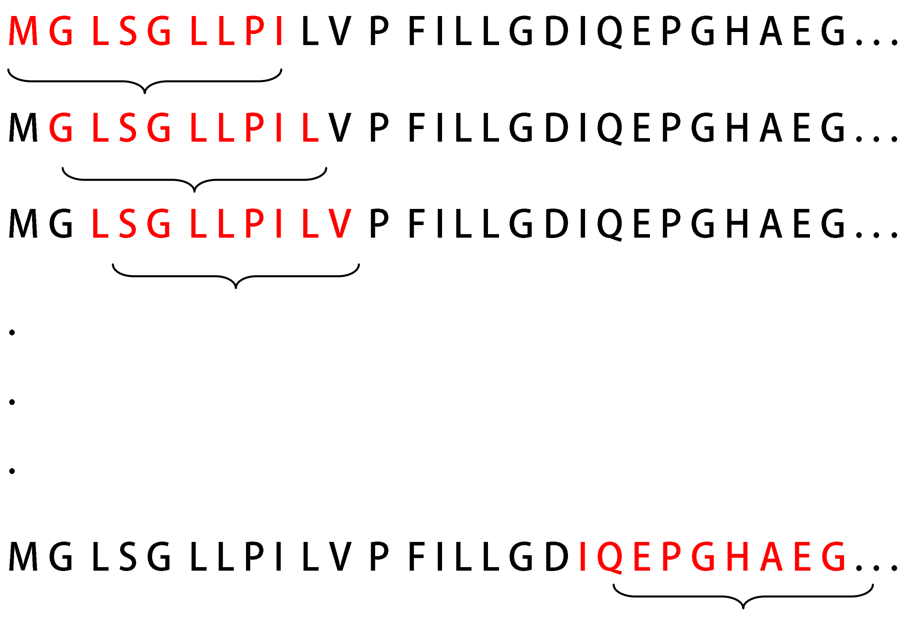
Both functions take as input one protein sequence at a time and one single numeric value for the peptide length. To compute peptides of multiple lengths, the functions should be called multiple times. For the next sections we assume that we have the sequence:

MGLSGLLPILVPILLGDIQEPGHAEGILGKPCPKIKVE

> p_nonspliced():

It returns the non-spliced peptides (catalyzed peptides - CP). In a sequence of N amino-acids there are $N - Nmer + 1$ catalyzed peptides. Each peptide starts from amino-acid AA_i where $i = 1, 2, 3, \dots, N - Nmer + 1$ and ends at amino-acid AA_j $j = Nmer, Nmer + 1,\ Nmer + 2,\ Nmer + 3 \dots \dots, Nmer$.

For example, in the first part of our sequence we get the illustrated 9mer peptides:


M G L S G L L P I L V P F I L L G D I Q E P G H A E G . . .
M G L S G L L P I L V P F I L L G D I Q E P G H A E G . . .
M G L S G L L P I L V P F I L L G D I Q E P G H A E G . . .
M G L S G L L P I L V P F I L L G D I Q E P G H A E G . . .

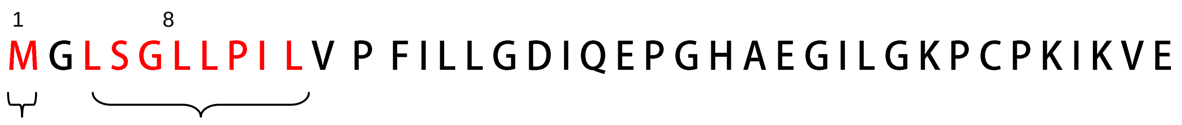
The output is a list of two character vectors: one with all non-spliced peptides and one with the corresponding amino-acids indexes in the parental sequence. The index for each peptide is denoted by “ i, j ”.

>p_spliced()

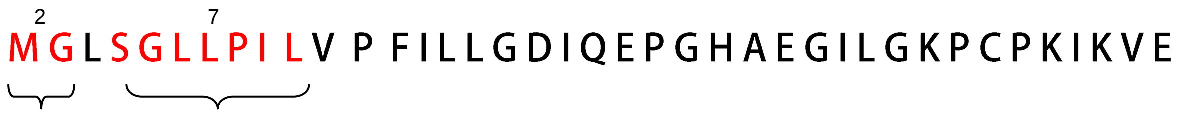
The spliced peptides are computed by creating all possible combinations between two non-continuous amino-acid fragments for the given $isl\ (min, max)$. Two given fragments are separated in the original sequence by at least $isl\ (min)$ and at most $isl\ (max)$ amino acids. To compute the first two peptides we start from the first amino-acid AA_1 in the protein sequence (first single AA fragment), skip $isl\ (min)$ amino-acids and merge it with $Nmer - 1$ continuous AA onward (second fragment). We get two Nmer spliced peptides; one by combining the fragments in normal order and one in reverse. To create the third and fourth peptide, we extend the first fragment by one amino-acid onward, i.e AA_1, AA_2, we skip $isl(min)$ amino-acids starting from $AA_3$and we merge it with Nmer-2 continuous AA onward

After all fragment combinations for the $isl\ (min)$ are reached within the sequence(from $AA_1\ to\ AA_N$, the fragmentation starts again from AA_1 skips $isl\ (min + 1)$ amino-acids and creates the second fragment with $Nmer - 1$ continuous AA onward. When all fragmentations for isl (min+1) and the starting point AA_1 are completed, the process is repeated starting from AA_2, and so forth.

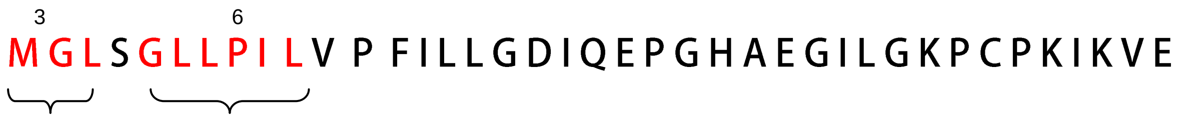
An illustrated example for $Nmer = 9$ and $isl\ = \ c(1, 25)$:


1
8
M G L S G L L P I L V P F I L L G D I Q E P G H A E G I L G K P C P K I K V E

$SP_1 = MLSGLLPIL\,,\quad SP_2 = LSGLLPILM$


2
7
M G L S G L L P I L V P F I L L G D I Q E P G H A E G I L G K P C P K I K V E

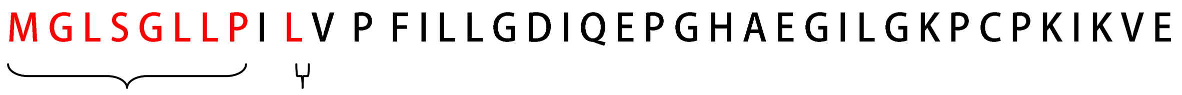
$SP_3 = MGSGLLPIL\,,\quad SP_4 = SGLLPILMG$


3
6
M G L S G L L P I L V P F I L L G D I Q E P G H A E G I L G K P C P K I K V E

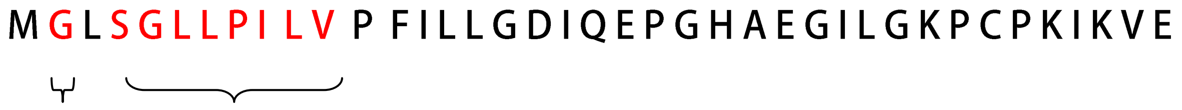
$SP_5 = MGLGLLPIL\,,\quad SP_6 = GLLPILMGL$

.

.


M G L S G L L P I L V P F I L L G D I Q E P G H A E G I L G K P C P K I K V E

$SP_{15} = MGLSGLLPL\,,\quad SP_{16} = LMGLSGLLP$

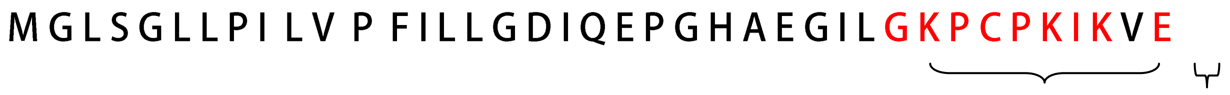

M G L S G L L P I L V P F I L L G D I Q E P G H A E G I L G K P C P K I K V E

$SP_{17} = MGLSGLLPL\,,\quad SP_{18} = LMGLSGLLP$ (first peptide with starting point AA_2)

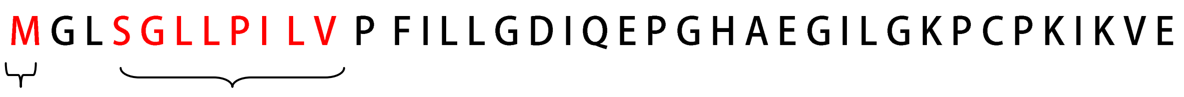
.

.

.


M G L S G L L P I L V P F I L L G D I Q E P G H A E G I L G K P C P K I K V E

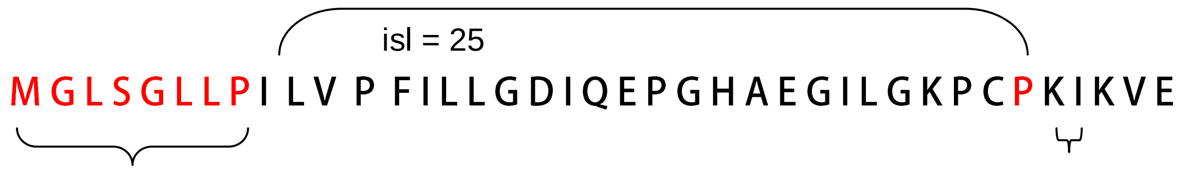
$SP_{479} = GKPCPKIKE\,,\quad SP_{480} = EGKPCPKIK$ (last peptides with $isl = 1$)


M G L S G L L P I L V P F I L L G D I Q E P G H A E G I L G K P C P K I K V E

$\boldsymbol{SP_{481}} = \boldsymbol{MSGLLPILV}$, $\boldsymbol{SP_{482}} = \boldsymbol{SGLLPILVM}$ (first peptides with $isl = 2$)

.

.

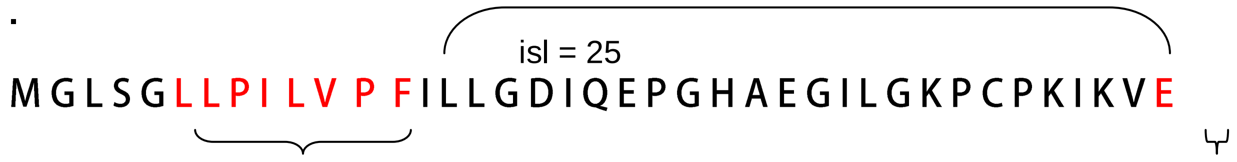

isl = 25
M G L S G L L P I L V P F I L L G D I Q E P G H A E G I L G K P C P K I K V E

$\boldsymbol{SP_{7119}} = \boldsymbol{MGLSGLLPP}$, $\boldsymbol{SP_{7120}} = \boldsymbol{PMGLSGLLP}$ (last peptides with starting point AA_1 and $isl = 25$)

.

.

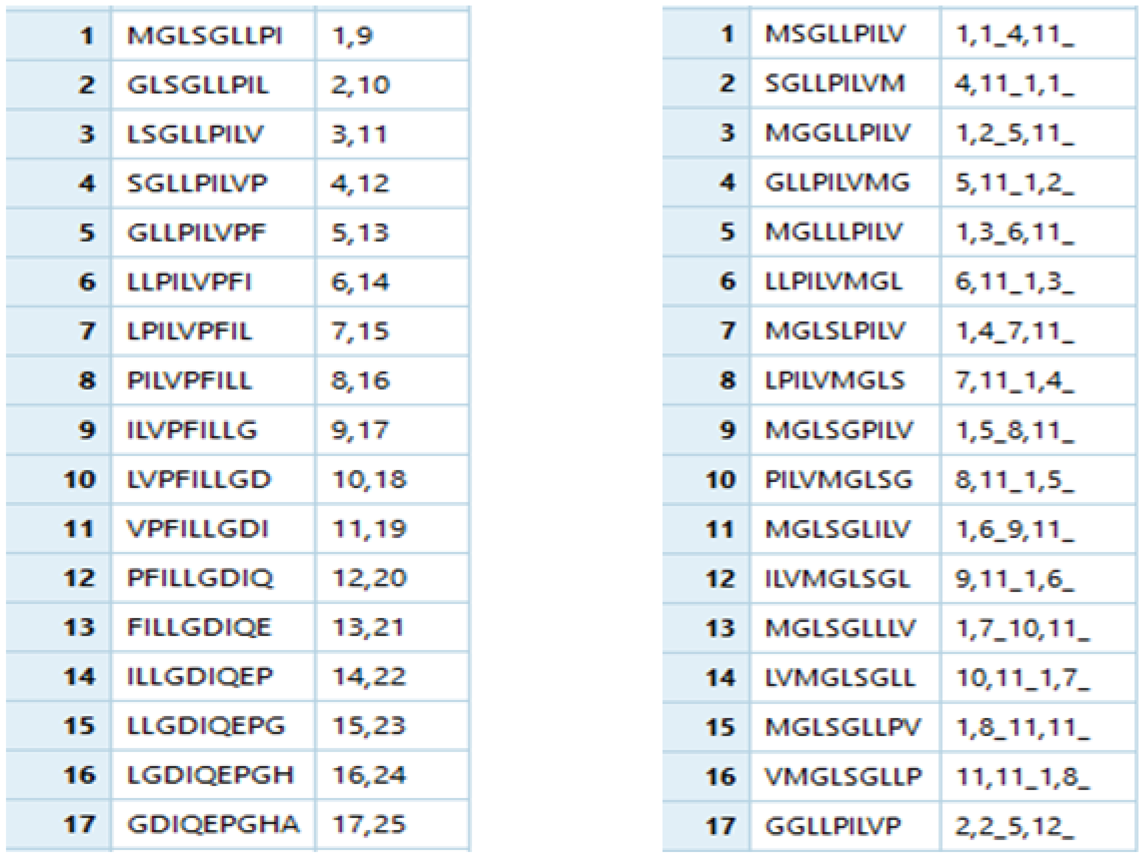
.


isl = 25
M G L S G L L P I L V P F I L L G D I Q E P G H A E G I L G K P C P K I K V E

$\boldsymbol{SP_{7199}} = \boldsymbol{LLPILVPFE}$, $\boldsymbol{SP_{7200}} = \boldsymbol{ELLPILVPF}$ (last peptides with isl=25, last in the sequence)

Again, the output is a list of two character vectors: one with all spliced peptides and one with indices. The index for each peptide is denoted as $i_1,j_1\_\ i_2,j_2$, where the first fragment starts at i_1 and ends at j_2 and the second fragment starts at i_2 and ends at j_2. For example, for the first spliced peptide SP_1 the index will be 1,1 _ 3,10, meaning that the first fragment starts and ends at amino-acid AA_1, and the second fragment starts at AA_3 and ends at AA_{10} .

Both functions are implemented in C++ and return a non-reduced database which may contain duplicates. As an extra step non-unique peptides should be removed manually and non-spliced should be removed from spliced database. For the next steps, lists should be translated into data frames of two columns (one with peptides, the other with indices) as shown in **figure 3.3**:



1	MGLSGLLPI	1,9
2	GLSGLLPIL	2,10
3	LSGLLPILV	3,11
4	SGLLPILVP	4,12
5	GLLPILVPF	5,13
6	LLPILVPFI	6,14
7	LPILVPFIL	7,15
8	PILVPFILL	8,16
9	ILVPFILLG	9,17
10	LVPFILLGD	10,18
11	VPFILLGDI	11,19
12	PFILLGDIQ	12,20
13	FILLGDIQE	13,21
14	ILLGDIQEP	14,22
15	LLGDIQEPG	15,23
16	LGDIQEPGH	16,24
17	GDIQEPGHA	17,25

1	MSGLLPILV	1,1_4,11_
2	SGLLPILVM	4,11_1,1_
3	MGGLLPILV	1,2_5,11_
4	GLLPILVMG	5,11_1,2_
5	MGLLLPILV	1,3_6,11_
6	LLPILVMGL	6,11_1,3_
7	MGLSLPILV	1,4_7,11_
8	LPILVMGLS	7,11_1,4_
9	MGLSGPILV	1,5_8,11_
10	PILVMGLSG	8,11_1,5_
11	MGLSGLILV	1,6_9,11_
12	ILVMGLSGL	9,11_1,6_
13	MGLSGLLLV	1,7_10,11_
14	LVMGLSGLL	10,11_1,7_
15	MGLSGLLPV	1,8_11,11_
16	VMGLSGLLP	11,11_1,8_
17	GGLLPILVP	2,2_5,12_

Figure 3.3 Output samples of non-spliced and spliced peptides

3.2.3 Filtering

As previously explained, the complete databases can become vast occupying large disc space. It is advised to reduce the database entries to only those which are actual candidates, if the experimental masses are available. To start the filtering process we call:

>mass_filter(D, masslist, tolerance)

Where:

- ***D*** is a data frame of the format shown in **figure 3.3** containing the peptides to be filtered
- ***masslist*** is a numeric vector containing all the ion masses found in the MS analysis
- ***tolerance*** is a numerical value in parts per million

The function computes the molecular weight (mw) for each peptide and a range with minimum and maximum weight according to tolerance. Tolerance of 3 ppm will result in the range($mw - mw * 1{,}5 * 10^{-6}$, $mw + mw * 1{,}5 * 10^{-6}$). This range is stored as a *minimum* and a *maximum* column in the D. If at least one mass entry from the *masslist* vector falls within a peptide's molecular weight range, this peptide shall remain in the database.

To efficiently implement that, the *masslist* and the peptide data frame *D* are sorted ascending according to molecular weight. The mass list is reduced to contain equal or greater values than the first element of the minimum MW column and equal or less values than the last element of maximum MW column. Now instead of searching each mass entry within the whole molecular weight column, we search until the queried mass in *masslist* is larger than the maximum weight of a peptide in the *D*.

Let's assume we have two indices: i for indexing the peptide dataframe rows and j for the *masslist* elements. Both indices start at 1.

Preprocessing:

i) Compute columns $mw, mw.min$ and $mw.max$ for each peptide in D
ii) Sort D according to mw ascending. Sort $masslist$ ascending
iii) Remove elements in $masslist$ if $j \leq mw.min_1$
iv) Remove elements in $masslist$ if $j \geq mw.max_n$

Now, by default, the following are always true:

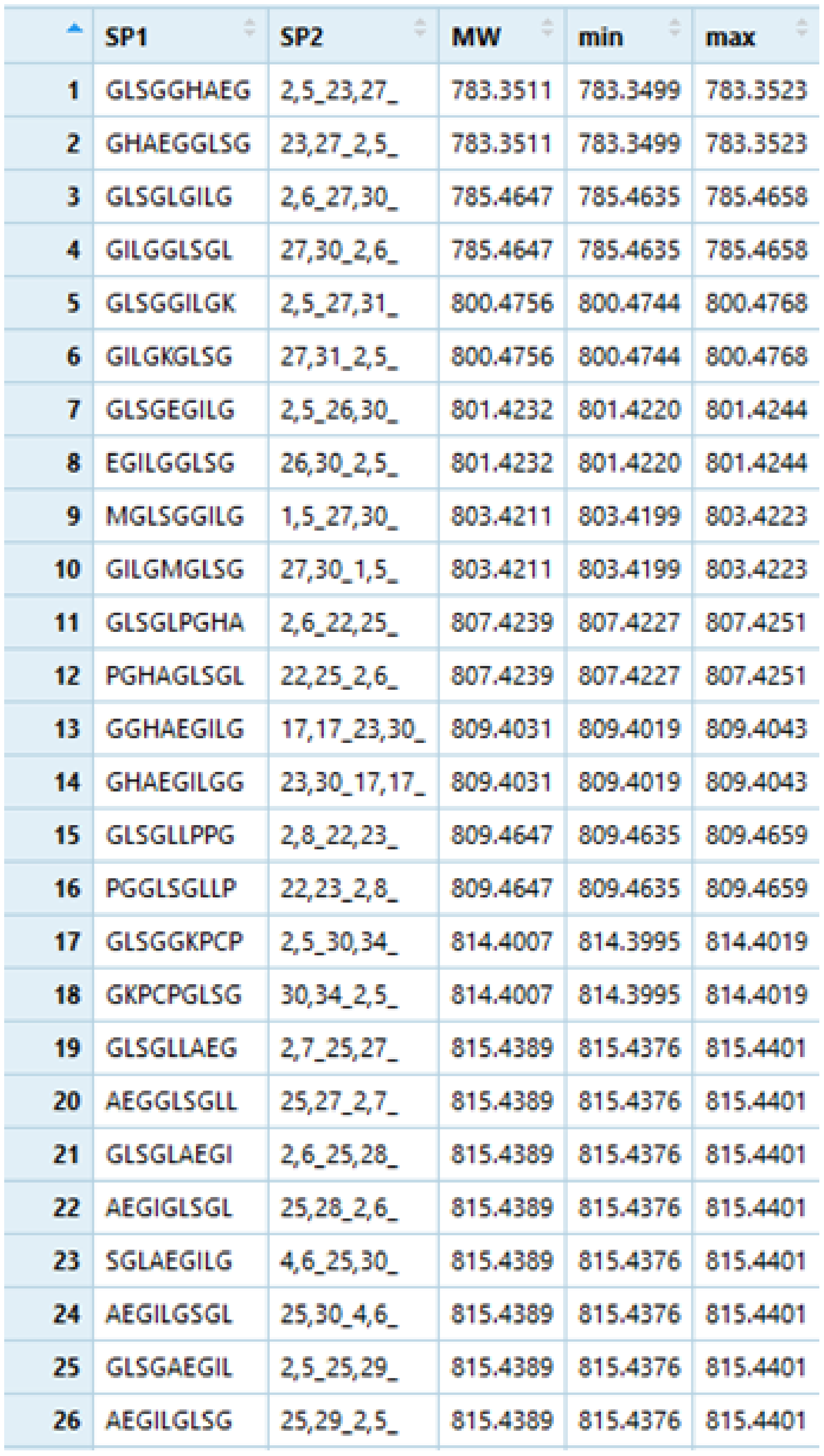
1. $mw.min_i \leq mw.min_{i+1}$
2. $mw.max_i \leq mw.max_{i+1}$
3. $j_1 \geq mw.min_1$

Then, the control flow for filtering is as below:

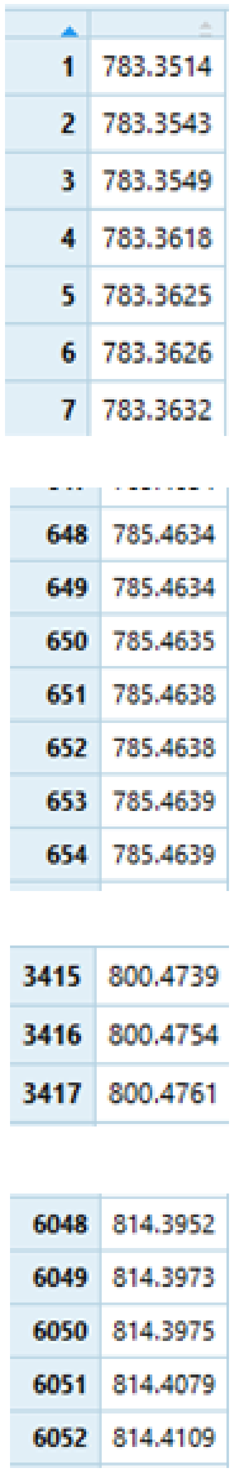
A) If the current j value is equal or less than the current i_{max} value we go through the following conditions, else we remove current i element and move to the next:

i) If j element is within min_i and max_i , the i element is kept in the database and we move to the next i (j remains the same) and start again at i).
ii) If j element is greater than max_i , the i element is removed from the database and we move to the next i (j remains the same) and start again at i).
iii) If, and only if, the j element is smaller than min_i then we move to the next j element and exit at A).

For example, let's assume that the first elements in the sorted peptide data frame (where SP1 column contains the theoretical peptides, SP2 the AA indexes, MW the computed molecular weight ,min-max the MW range) and mass list are as shown in **Figure 3.4** and represented by i and j indexes respectively.



	SP1	SP2	MW	min	max
1	GLSGGHAEG	2,5_23,27_	783.3511	783.3499	783.3523
2	GHAEGGLSG	23,27_2,5_	783.3511	783.3499	783.3523
3	GLSGLGILG	2,6_27,30_	785.4647	785.4635	785.4658
4	GILGGLSGL	27,30_2,6_	785.4647	785.4635	785.4658
5	GLSGGILGK	2,5_27,31_	800.4756	800.4744	800.4768
6	GILGKGLSG	27,31_2,5_	800.4756	800.4744	800.4768
7	GLSGEGILG	2,5_26,30_	801.4232	801.4220	801.4244
8	EGILGGLSG	26,30_2,5_	801.4232	801.4220	801.4244
9	MGLSGGILG	1,5_27,30_	803.4211	803.4199	803.4223
10	GILGMGLSG	27,30_1,5_	803.4211	803.4199	803.4223
11	GLSGLPGHA	2,6_22,25_	807.4239	807.4227	807.4251
12	PGHAGLSGL	22,25_2,6_	807.4239	807.4227	807.4251
13	GGHAEGILG	17,17_23,30_	809.4031	809.4019	809.4043
14	GHAEGILGG	23,30_17,17_	809.4031	809.4019	809.4043
15	GLSGLLPPG	2,8_22,23_	809.4647	809.4635	809.4659
16	PGGLSGLLP	22,23_2,8_	809.4647	809.4635	809.4659
17	GLSGGKPCP	2,5_30,34_	814.4007	814.3995	814.4019
18	GKPCPGLSG	30,34_2,5_	814.4007	814.3995	814.4019
19	GLSGLLAEG	2,7_25,27_	815.4389	815.4376	815.4401
20	AEGGLSGLL	25,27_2,7_	815.4389	815.4376	815.4401
21	GLSGLAEGI	2,6_25,28_	815.4389	815.4376	815.4401
22	AEGIGLSGL	25,28_2,6_	815.4389	815.4376	815.4401
23	SGLAEGILG	4,6_25,30_	815.4389	815.4376	815.4401
24	AEGILGSGL	25,30_4,6_	815.4389	815.4376	815.4401
25	GLSGAEGIL	2,5_25,29_	815.4389	815.4376	815.4401
26	AEGILGLSG	25,29_2,5_	815.4389	815.4376	815.4401



1	783.3514
2	783.3543
3	783.3549
4	783.3618
5	783.3625
6	783.3626
7	783.3632
648	785.4634
649	785.4634
650	785.4635
651	785.4638
652	785.4638
653	785.4639
654	785.4639
3415	800.4739
3416	800.4754
3417	800.4761
6048	814.3952
6049	814.3973
6050	814.3975
6051	814.4079
6052	814.4109

Figure 3.4 Samples of data frame D and vector $masslist$ for the filtering process

The element j_1 in mass list is $783{,}3514$. It is within the peptide i_1 MW range, so the peptide is marked as “K” (Keep) and move to i_2. Due to sorting, whatever the current peptide is, the next peptide in the list will have either the same MW value or greater, thus its min, max range will be either the same or shifted to the right in the numerical scale.

The i_2 element is equal to i_1 so it is also a “K” and we move to i_3. The j_1 is lower than the $mw.min$ and $mw.max$ of i_3 so we need to move to j_2 which is still lower than i_3. This also applies for the next rows so we keep moving in j column until we arrive at j_{650} . This is within the MW range of i_3 so we mark peptide i_3 as a “K”. Same applies for i_4.

For element i_5 we need to skip many rows of j to get to a j element of similar value. We keep moving in j column until we get to j_{3416} which is within i_5 MW range, we mark i_5 as a “K” and we keep moving on in i list

When we get to i_{17} and as we are still increasing the j (condition A.iii.) , we get to j_{6051} which is greater than the $mw.min$ but also greater than $mw.max$ of i_{17} . This peptide is, thus, not represented in the masslist so we mark it as an “R” (remove) and move to i_{18} which equal to i_{17} and removed as well.

Getting to i_{19} we enter again in condition iii) and we start moving on j list until we either get a match or j exceeds a $mw.\max$ i element so we start moving in the i list.

This way it’s not needed to search the whole mass list against each individual peptide in the database. Instead, we go through both lists simultaneously, saving a lot of time.

This part of the algorithm is implemented in C++ for optimized speed.

3.2.4 Output in protein format

Before calling the main function that returns the database in the desired format, we need to estimate the distribution of the current proteome database. We call:
>Distrifit(data, d)

Where:

- ***Data*** is the input proteome database in FASTA format
- ***d*** is a string variable indicating the distribution to be fitted, “lognormal” or “gamma”

This returns a vector of the distribution parameters i.e. mean and variance in the case of a log-normal distribution, k and θ in the case of a gamma distribution. This step, however, can be performed manually by the user who should choose a distribution that best describes the data. In any case, this step is only complementary and reduction in accuracy of fitting will not alter the results of the downstream analysis in any way.

Then the format conversion process can start. In this step, the peptides are stitched together into peptide-blocks to resemble proteins. After obtaining a peptide data frame (as in figure 3.1) we call:

>create_blocks(x, title, pep.type,fit, nmer)

Where:

- ***x*** is a dataframe which has in the first column all the peptides to be stitched
- ***title*** is the title of the original protein, needed for annotation
- ***pep.type*** is the peptide type (spliced or non-spliced), needed for annotation
- ***fit*** is the output of *Distrfit()* function. A list containing a vector with a character element which indicates the distribution, and a numerical vector of the distribution parameters from which the protein lengths will be sampled.
- ***Nmer*** is the peptide sequence length.

A random log normal or gamma generator is used to sample one thousand values from the distribution of choice. Each value corresponds to the length of the current peptide-block. For example, if the first value in the vector is $N_1 = 900$ then $\frac{900}{Nmer} = \frac{900}{9} = 100$ peptides will be used in the first peptide-block. If the peptide list contains 1000 peptides then the first 100 will be stitched together leaving behind 900 peptides. Thus, for the second peptide-block the index starts from peptide 101 and uses N_2 peptides, i.e the second value in the sampled vector. If the peptide list is so large that more that 1000 blocks are needed thus exceeding the length of the vector, then the index of the sampled vector starts again at N_1. This, however, does not happen often so it is preferable to keep the sample vector reasonably short than allocating useless elements that occupy RAM and slow down the process.

Let's assume we have already called the *p_spliced()* function which returned a list of spliced peptides and transformed it into a data frame.
After calling the *create_blocks()* function the returned list includes all peptide-blocks composed by spliced peptides derived from *Protein1,* as shown in **Figure 3.5** .

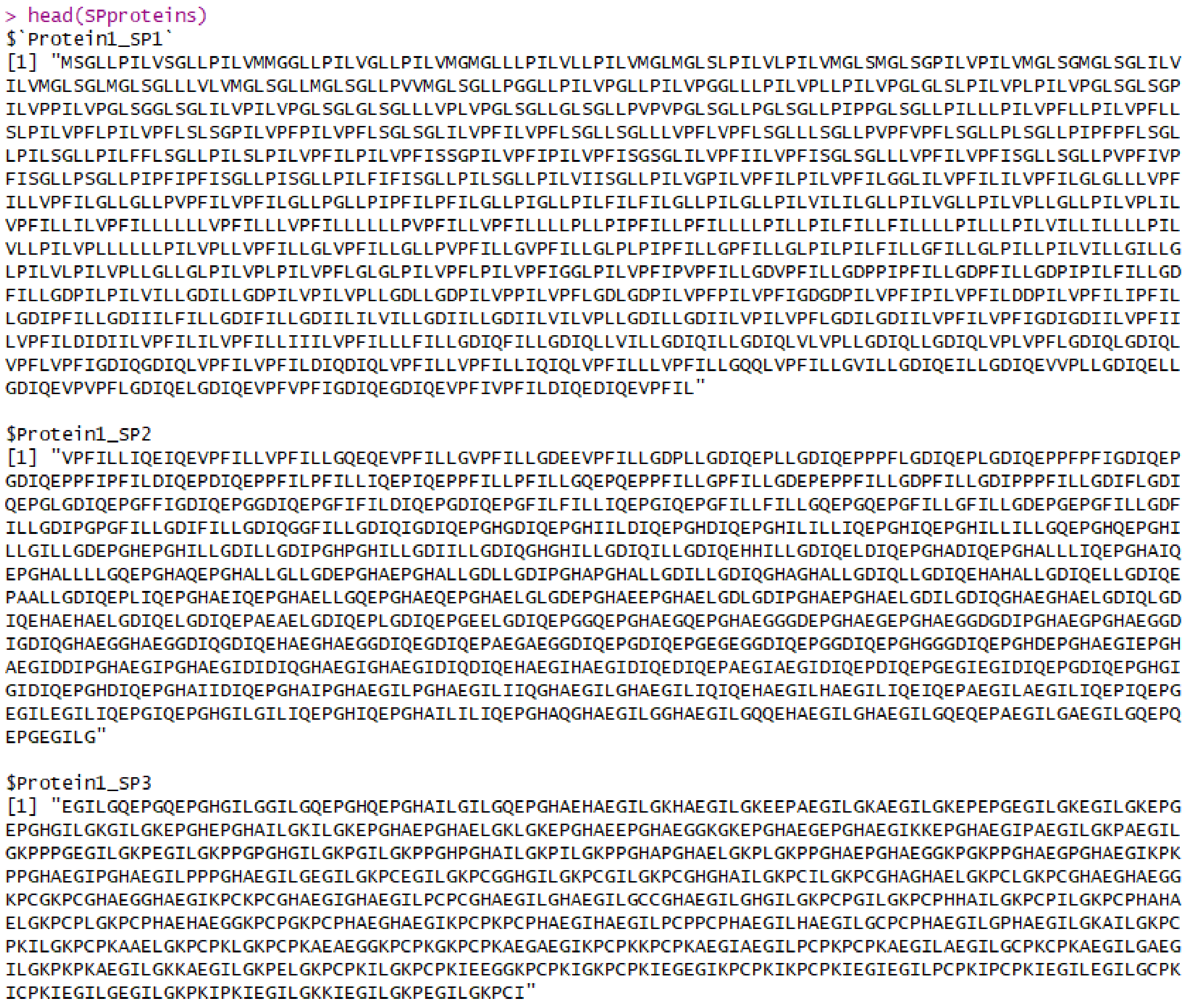
3.2.5 Decoy database

To obtain a decoy database of peptides from the current protein, the function *create_decoy()* is called :

>create_decoy(x, title, pep.type, method)

Where:

- ***x*** is a character vector of peptides or a list of peptide sequences in peptide-block format.
- ***title*** is the title of the protein for annotation
- ***pep.type*** is the type of peptides (spliced or non-spliced) for annotation
- ***method*** is a string variable indicating the decoy method to be used. Possible values are "rev" for reverse sequences or "shf" for shuffled sequences



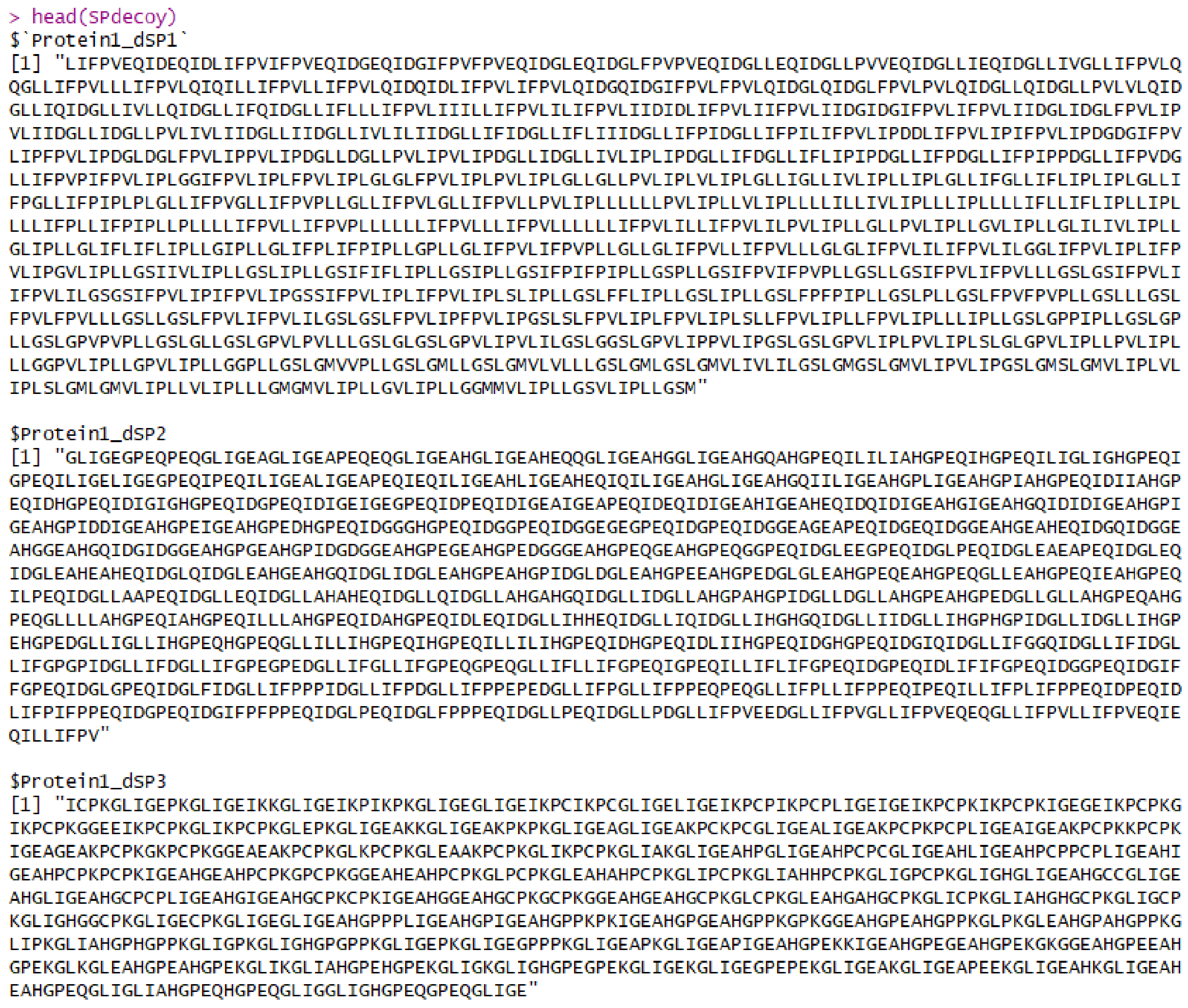
```
> head(SPproteins)
$`Protein1_SP1`
[1] "MSGLLPILVSGLLPILVMMGGLLPILVGLLPILVMGMGLLLPILVLLPILVMGLMGLSLPILVLPILVMGLSMGLSGPILVPILVMGLSGMGLSGLILV
ILVMGLSGLMGLSGLLLVLVMGLSGLLMGLSGLLPVVMGLSGLLPGGLLPILVPGLLPILVPGGLLLPILVPLLPILVPGLGLSLPILVPLPILVPGLSGLSGP
ILVPPILVPGLSGGLSGLILVPILVPGLSGLGLSGLLLVPLVPGLSGLLGLSGLLPVPVPGLSGLLPGLSGLLPIPPGLSGLLPILLLPILVPFLLPILVPFLL
SLPILVPFLPILVPFLSLSGPILVPFPILVPFLSGLSGLILVPFILVPFLSGLLSGLLLVPFLVPFLSGLLLSGLLPVPFVPFLSGLLPLSGLLPIPFPFLSGL
LPILSGLLPILFFLSGLLPILSLPILVPFILPILVPFISSGPILVPFIPILVPFISGSGLILVPFIILVPFISGLSGLLLVPFILVPFISGLLSGLLPVPFIVP
FISGLLPSGLLPIPFIPFISGLLPISGLLPILFIFISGLLPILSGLLPILVIISGLLPILVGPILVPFILPILVPFILGGLILVPFILILVPFILGLGLLLVPF
ILLVPFILGLLGLLPVPFILVPFILGLLPGLLPIPFILPFILGLLPIGLLPILFILFILGLLPILGLLPILVILILGLLPILVGLLPILVPLLGLLPILVPLIL
VPFILLILVPFILLLLLLVPFILLLVPFILLLLLLPVPFILLVPFILLLLPLLPIPFILLPFILLLLPILLPILFILLFILLLLPILLLPILVILLILLLLPIL
VLLPILVPLLLLLLPILVPLLVPFILLGLVPFILLGLLPVPFILLGVPFILLGLPLPIPFILLGPFILLGLPILPILFILLGFILLGLPILLPILVILLGILLG
LPILVLPILVPLLGLLGLPILVPLPILVPFLGLGLPILVPFLPILVPFIGGLPILVPFIPVPFILLGDVPFILLGDPPIPFILLGDPFILLGDPIPILFILLGD
FILLGDPILPILVILLGDILLGDPILVPILVPLLGDLLGDPILVPPILVPFLGDLGDPILVPFPILVPFIGDGDPILVPFIPILVPFILDDPILVPFILIPFIL
LGDIPFILLGDIIILFILLGDIFILLGDIILILVILLGDIILLGDIILVILVPLLGDILLGDIILVPILVPFLGDILGDIILVPFILVPFIGDIGDIILVPFII
LVPFILDIDIILVPFILILVPFILLIIILVPFILLLFILLGDIQFILLGDIQLLVILLGDIQILLGDIQLVLVPLLGDIQLLGDIQLVPLVPFLGDIQLGDIQL
VPFLVPFIGDIQGDIQLVPFILVPFILDIQDIQLVPFILLVPFILLIQIQLVPFILLLVPFILLGQQLVPFILLGVILLGDIQEILLGDIQEVVPLLGDIQELL
GDIQEVPVPFLGDIQELGDIQEVPFVPFIGDIQEGDIQEVPFIVPFILDIQEDIQEVPFIL"

$Protein1_SP2
[1] "VPFILLIQEIQEVPFILLVPFILLGQEQEVPFILLGVPFILLGDEEVPFILLGDPLLGDIQEPLLGDIQEPPPFLGDIQEPLGDIQEPPFPFIGDIQEP
GDIQEPPFIPFILDIQEPDIQEPPFILPFILLIQEPIQEPPFILLPFILLGQEPQEPPFILLGPFILLGDEPEPPFILLGDPFILLGDIPPPFILLGDIFLGDI
QEPGLGDIQEPGFFIGDIQEPGGDIQEPGFIFILDIQEPGDIQEPGFILFILLIQEPGIQEPGFILLFILLGQEPGQEPGFILLGFILLGDEPGEPGFILLGDF
ILLGDIPGPGFILLGDIFILLGDIQGGFILLGDIQIGDIQEPGHGDIQEPGHIILDIQEPGHDIQEPGHILILLIQEPGHIQEPGHILLILLGQEPGHQEPGHI
LLGILLGDEPGHEPGHILLGDILLGDIPGHPGHILLGDIILLGDIQGHGHILLGDIQILLGDIQEHHILLGDIQELDIQEPGHADIQEPGHALLLIQEPGHAIQ
EPGHALLLLGQEPGHAQEPGHALLGLLGDEPGHAEPGHALLGDLLGDIPGHAPGHALLGDILLGDIQGHAGHALLGDIQLLGDIQEHAHALLGDIQELLGDIQE
PAALLGDIQEPLIQEPGHAEIQEPGHAELLGQEPGHAEQEPGHAELGLGDEPGHAEEPGHAELGDLGDIPGHAEPGHAELGDILGDIQGHAEGHAELGDIQLGD
IQEHAEHAELGDIQELGDIQEPAEAELGDIQEPLGDIQEPGEELGDIQEPGGQEPGHAEGQEPGHAEGGGDEPGHAEGEPGHAEGGDGDIPGHAEGPGHAEGGD
IGDIQGHAEGGHAEGGDIQGDIQEHAEGHAEGGDIQEGDIQEPAEGAEGGDIQEPGDIQEPGEGEGGDIQEPGGDIQEPGHGGGDIQEPGHDEPGHAEGIEPGH
AEGIDDIPGHAEGIPGHAEGIDIDIQGHAEGIGHAEGIDIQDIQEHAEGIHAEGIDIQEDIQEPAEGIAEGIDIQEPDIQEPGEGIEGIDIQEPGDIQEPGHGI
GIDIQEPGHDIQEPGHAIIDIQEPGHAIPGHAEGILPGHAEGILIIQGHAEGILGHAEGILIQIQEHAEGILHAEGILIQEIQEPAEGILAEGILIQEPIQEPG
EGILEGILIQEPGIQEPGHGILGILIQEPGHIQEPGHAILILIQEPGHAQGHAEGILGGHAEGILGQQEHAEGILGHAEGILGQEQEPAEGILGAEGILGQEPQ
EPGEGILG"

$Protein1_SP3
[1] "EGILGQEPGQEPGHGILGGILGQEPGHQEPGHAILGILGQEPGHAEHAEGILGKHAEGILGKEEPAEGILGKAEGILGKEPEPGEGILGKEGILGKEPG
EPGHGILGKGILGKEPGHEPGHAILGKILGKEPGHAEPGHAELGKLGKEPGHAEEPGHAEGGKGKEPGHAEGEPGHAEGIKKEPGHAEGIPAEGILGKPAEGIL
GKPPPGEGILGKPEGILGKPPGPGHGILGKPGILGKPPGHPGHAILGKPILGKPPGHAPGHAELGKPLGKPPGHAEPGHAEGGKPGKPPGHAEGPGHAEGIKPK
PPGHAEGIPGHAEGILPPPGHAEGILGEGILGKPCEGILGKPCGGHGILGKPCGILGKPCGHGHAILGKPCILGKPCGHAGHAELGKPCLGKPCGHAEGHAEGG
KPCGKPCGHAEGGHAEGIKPCKPCGHAEGIGHAEGILPCPCGHAEGILGHAEGILGCCGHAEGILGHGILGKPCPGILGKPCPHHAILGKPCPILGKPCPHAHA
ELGKPCPLGKPCPHAEHAEGGKPCPGKPCPHAEGHAEGIKPCPKPCPHAEGIHAEGILPCPPCPHAEGILHAEGILGCPCPHAEGILGPHAEGILGKAILGKPC
PKILGKPCPKAAELGKPCPKLGKPCPKAEAEGGKPCPKGKPCPKAEGAEGIKPCPKKPCPKAEGIAEGILPCPKPCPKAEGILAEGILGCPKCPKAEGILGAEG
ILGKPKPKAEGILGKKAEGILGKPELGKPCPKILGKPCPKIEEGGKPCPKIGKPCPKIEGEGIKPCPKIKPCPKIEGIEGILPCPKIPCPKIEGILEGILGCPK
ICPKIEGILGEGILGKPKIPKIEGILGKKIEGILGKPEGILGKPCI"
```

Figure 3.5: Output to peptide-blocks format for *Protein1*

The output is either in peptide-blocks format or a vector of peptides depending on the input format. Two methods are available to construct a decoy: reverse and shuffled (as described in section 2.3).

For example, let's assume we have already called the *create_blocks()* function which returned those peptides in protein format as shown in Figure 3.5 . To get the reverse-decoy sequences that correspond to *Protein1*, we need to call *create_decoy(x, title="Protein1", pep.type= "SP", method= "rev")* .The **Figure 3.6** illustrates the list output when reversed method is chosen.

Each protein's spliced peptides produce a separate file of decoy sequences. For *Protein1* a FASTA file with title "Protein1_dSP.txt" is saved. Each decoy peptide-block is annotated with an index number (1, 2, 3 etc).



```
> head(SPdecoy)
$`Protein1_dSP1`
[1] "LIFPVEQIDEQIDLIFPVIFPVEQIDGEQIDGIFPVFPVEQIDGLEQIDGLFPVPVEQIDGLLEQIDGLLPVVEQIDGLLIEQIDGLLIVGLLIFPVLQ
QGLLIFPVLLLIFPVLQIQILLIFPVLLIFPVLQIDQIDLIFPVLIFPVLQIDGQIDGIFPVLFPVLQIDGLQIDGLFPVLPVLQIDGLLQIDGLLPVLVLQID
GLLIQIDGLLIVLLQIDGLLIFQIDGLLIFLLLIFPVLIIILLIFPVLILIFPVLIIDIDLIFPVLIIFPVLIIDGIDGIFPVLIFPVLIIDGLIDGLFPVLIP
VLIIDGLLIDGLLPVLIVLIIDGLLIIDGLLIVLILIIDGLLIFIDGLLIFLIIIDGLLIFPIDGLLIFPILIFPVLIPDDLIFPVLIPIFPVLIPDGDGIFPV
LIPFPVLIPDGLDGLFPVLIPPVLIPDGLLDGLLPVLIPVLIPDGLLIDGLLIVLIPLIPDGLLIFDGLLIFLIPIPDGLLIFPDGLLIFPIPPDGLLIFPVDG
LLIFPVPIFPVLIPLGGIFPVLIPLFPVLIPLGLGLFPVLIPLPVLIPLGLLGLLPVLIPLVLIPLGLLIGLLIVLIPLLIPLGLLIFGLLIFLIPLIPLGLLI
FPGLLIFPIPLPLGLLIFPVGLLIFPVPLLGLLIFPVLGLLIFPVLLPVLIPLLLLLLPVLIPLLVLIPLLLLILLIVLIPLLLIPLLLLIFLLIFLIPLLIPL
LLLIFPLLIFPIPLLPLLLLIFPVLLIFPVPLLLLLLIFPVLLLIFPVLLLLLLIFPVLILLIFPVLILPVLIPLLGLLPVLIPLLGVLIPLLGLILIVLIPLL
GLIPLLGLIFLIFLIPLLGIPLLGLIFPLIFPIPLLGPLLGLIFPVLIFPVPLLGLLGLIFPVLLIFPVLLLGLGLIFPVLILIFPVLILGGLIFPVLIPLIFP
VLIPGVLIPLLGSIIVLIPLLGSLIPLLGSIFIFLIPLLGSIPLLGSIFPIFPIPLLGSPLLGSIFPVIFPVPLLGSLLGSIFPVLIFPVLLLGSLGSIFPVLI
IFPVLILGSGSIFPVLIPIFPVLIPGSSIFPVLIPLIFPVLIPLSLIPLLGSLFFLIPLLGSLIPLLGSLFPFPIPLLGSLPLLGSLFPVFPVPLLGSLLLGSL
FPVLFPVLLLGSLLGSLFPVLIFPVLILGSLGSLFPVLIPFPVLIPGSLSLFPVLIPLFPVLIPLSLLFPVLIPLLFPVLIPLLLIPLLGSLGPPIPLLGSLGP
LLGSLGPVPVPLLGSLGLLGSLGPVLPVLLLGSLGLGSLGPVLIPVLILGSLGGSLGPVLIPPVLIPGSLGSLGPVLIPLPVLIPLSLGLGPVLIPLLPVLIPL
LLGGPVLIPLLGPVLIPLLGGPLLGSLGMVVPLLGSLGMLLGSLGMVLVLLLGSLGMLGSLGMVLIVLILGSLGMGSLGMVLIPVLIPGSLGMSLGMVLIPLVL
IPLSLGMLGMVLIPLLVLIPLLLGMGMVLIPLLGVLIPLLGGMMVLIPLLGSVLIPLLGSM"

$Protein1_dSP2
[1] "GLIGEGPEQPEQGLIGEAGLIGEAPEQEQGLIGEAHGLIGEAHEQQGLIGEAHGGLIGEAHGQAHGPEQILILIAHGPEQIHGPEQILIGLIGHGPEQI
GPEQILIGELIGEGPEQIPEQILIGEALIGEAPEQIEQILIGEAHLIGEAHEQIQILIGEAHGLIGEAHGQIILIGEAHGPLIGEAHGPIAHGPEQIDIIAHGP
EQIDHGPEQIDIGIGHGPEQIDGPEQIDIGEIGEGPEQIDPEQIDIGEAIGEAPEQIDEQIDIGEAHIGEAHEQIDQIDIGEAHGIGEAHGQIDIDIGEAHGPI
GEAHGPIDDIGEAHGPEIGEAHGPEDHGPEQIDGGGHGPEQIDGGPEQIDGGEGEGPEQIDGPEQIDGGEAGEAPEQIDGEQIDGGEAHGEAHEQIDGQIDGGE
AHGGEAHGQIDGIDGGEAHGPGEAHGPIDGDGGEAHGPEGEAHGPEDGGGEAHGPEQGEAHGPEQGGPEQIDGLEEGPEQIDGLPEQIDGLEAEAPEQIDGLEQ
IDGLEAHEAHEQIDGLQIDGLEAHGEAHGQIDGLIDGLEAHGPEAHGPIDGLDGLEAHGPEEAHGPEDGLGLEAHGPEQEAHGPEQGLLEAHGPEQIEAHGPEQ
ILPEQIDGLLAAPEQIDGLLEQIDGLLAHAHEQIDGLLQIDGLLAHGAHGQIDGLLIDGLLAHGPAHGPIDGLLDGLLAHGPEAHGPEDGLLGLLAHGPEQAHG
PEQGLLLLAHGPEQIAHGPEQILLLAHGPEQIDAHGPEQIDLEQIDGLLIHHEQIDGLLIQIDGLLIHGHGQIDGLLIIDGLLIHGPHGPIDGLLIDGLLIHGP
EHGPEDGLLIGLLIHGPEQHGPEQGLLILLIHGPEQIHGPEQILLILIHGPEQIDHGPEQIDLIIHGPEQIDGHGPEQIDGIQIDGLLIFGGQIDGLLIFIDGL
LIFGPGPIDGLLIFDGLLIFGPEGPEDGLLIFGLLIFGPEQGPEQGLLIFLLIFGPEQIGPEQILLIFLIFGPEQIDGPEQIDLIFIFGPEQIDGGPEQIDGIF
FGPEQIDGLGPEQIDGLFIDGLLIFPPPIDGLLIFPDGLLIFPPEPEDGLLIFPGLLIFPPEQPEQGLLIFPLLIFPPEQIPEQILLIFPLIFPPEQIDPEQID
LIFPIFPPEQIDGPEQIDGIFPFPPEQIDGLPEQIDGLFPPPEQIDGLLPEQIDGLLPDGLLIFPVEEDGLLIFPVGLLIFPVEQEQGLLIFPVLLIFPVEQIE
QILLIFPV"

$Protein1_dSP3
[1] "ICPKGLIGEPKGLIGEIKKGLIGEIKPIKPKGLIGEGLIGEIKPCIKPCGLIGELIGEIKPCPIKPCPLIGEIGEIKPCPKIKPCPKIGEGEIKPCPKG
IKPCPKGGEEIKPCPKGLIKPCPKGLEPKGLIGEAKKGLIGEAKPKPKGLIGEAGLIGEAKPCKPCGLIGEALIGEAKPCPKPCPLIGEAIGEAKPCPKKPCPK
IGEAGEAKPCPKGKPCPKGGEAEAKPCPKGLKPCPKGLEAAKPCPKGLIKPCPKGLIAKGLIGEAHPGLIGEAHPCPCGLIGEAHLIGEAHPCPPCPLIGEAHI
GEAHPCPKPCPKIGEAHGEAHPCPKGPCPKGGEAHEAHPCPKGLPCPKGLEAHAHPCPKGLIPCPKGLIAHHPCPKGLIGPCPKGLIGHGLIGEAHGCCGLIGE
AHGLIGEAHGCPCPLIGEAHGIGEAHGCPKCPKIGEAHGGEAHGCPKGCPKGGEAHGEAHGCPKGLCPKGLEAHGAHGCPKGLICPKGLIAHGHGCPKGLIGCP
KGLIGHGGCPKGLIGECPKGLIGEGLIGEAHGPPPLIGEAHGPIGEAHGPPKPKIGEAHGPGEAHGPPKGPKGGEAHGPEAHGPPKGLPKGLEAHGPAHGPPKG
LIPKGLIAHGPHGPPKGLIGPKGLIGHGPGPPKGLIGEPKGLIGEGPPPKGLIGEAPKGLIGEAPIGEAHGPEKKIGEAHGPEGEAHGPEKGKGGEAHGPEEAH
GPEKGLKGLEAHGPEAHGPEKGLIKGLIAHGPEHGPEKGLIGKGLIGHGPEGPEKGLIGEKGLIGEGPEPEKGLIGEAKGLIGEAPEEKGLIGEAHKGLIGEAH
EAHGPEQGLIGLIAHGPEQHGPEQGLIGGLIGHGPEQGPEQGLIGE"
```

Figure 3.6 Decoy sequences of spliced peptides when peptide-blocks format is used

3.2.6 Output files in automated process

Output files are automatically saved only in the automated process. If functions are used separately the user needs to save the database in the desired format manually.

Every file saved corresponds to one and only one sequence in the input list provided. By default, each protein’s spliced peptides database (spliced, non-spliced, decoy) is saved as an .rds file with title “sp_protein.rds” where *protein* is the unique Uniprot name. Those files are needed for the step 3.1.8 where the hits are searched against each database.

If output to blocks is chosen, each database is also saved in FASTA format as mentioned in section 3.1.5 and 3.1.6. For a combined search method (as described in section 2.3.1) all files of decoy proteins and all files of target databases need to be concatenated together in one text file.

3.2.7 Searching the hits

This step requires that an MS analysis software of choice has been used to match the database peptides. All the peptides that were found as hits should be included in one vector.
To search the candidate peptides against all the databases, we call:

>find_hits(x, data, DB, Nmer)

Where:

- ***x*** is a character vector containing the peptides (each element is a peptide)
- ***data*** is the exact same Uniprot file in FASTA format that was used to create the database
- ***DB*** a character value or character vector indicating which database should be searched. Possible values are “SP” for spliced peptide data base and “NSP” for non-spliced. For each database both target and decoy is searched.
- ***Nmer*** is a vector with the minimum and maximum Nmer. For example for Nmer= c(9,10) the peptide-hits will be searched against 9mer and 10mer databases.

This returns a data frame of four columns: the first column contains the input peptides, the second the database that each peptide was found (spliced or non-spliced), the third indicates if it was in the target or the decoy and the fourth the indexes of the amino-acids in the parental protein. Obviously the fourth column has no value for the peptides that were found in decoy databases. Peptides that were not found anywhere will have NA values in all columns.

4. BENCHMARKING AND SYSTEM SPECIFICATION

4.1 System information

In this chapter we present a rough estimation of the execution time for each heavily computational task in the algorithm. The system on which the algorithm was tested is listed below:

Operating system:	Windows 10
RStudio Version:	1.1.463
R Version:	3.5.1 (2018-07-02)
CPU:	AMD Ryzen 5 2400g, 3600 MHz
RAM:	16 GB, 3200 MHz

4.2 Benchmarking: Settings and results

All tests are performed in serial and parallel mode for a group of 1010 proteins which is equal to 1/20 of the total human proteome. For parallel processing we use three physical CPU cores and six logical. The group of proteins follows the original proteome's length distribution, which was found to be lognormal, as shown in **figure 4.1** (excluding two unusually large proteins: Q8WZ42|TITIN_HUMAN and Q8WXI7|MUC16HUMAN of 34350 and 14507 amino acids respectively). This way we ensure that for every additional 1010 proteins group that follows the same lognormal distribution, the execution time will increase linearly. Longer lengths of proteins increase the runtime also linearly. The automated process and all separate functions are benchmarked. We also test different lengths of proteins to illustrate the linear relationship between length and execution time. Additionally, we benchmark the speed gain for every additional cpu core used in parallelization.

It should be noted that even though virtually any CPU can run the algorithm, for increased ranges of intervening sequences' lengths more RAM is mandatory. Here, we test on a conventional computer built of 16 GB RAM. The maximum range of intervening sequence that can be comfortably handled at once is [1-100] amino-acids. However, one can run the algorithm multiple times using different ranges when less RAM is available. For example, a range of [1-100] can be separated in two or four rounds of [1-25], [26-50], [51-75] and [76-100] taking virtually the same execution time. It is shown in the following plots that intervening sequence length and execution time are almost perfectly linearly correlated.

4.2.1 Automated process

We first benchmark the automated process where all functions are utilized, for 9mers and different ranges of intervening sequence lengths. Execution time is almost identical for any peptide length selected.

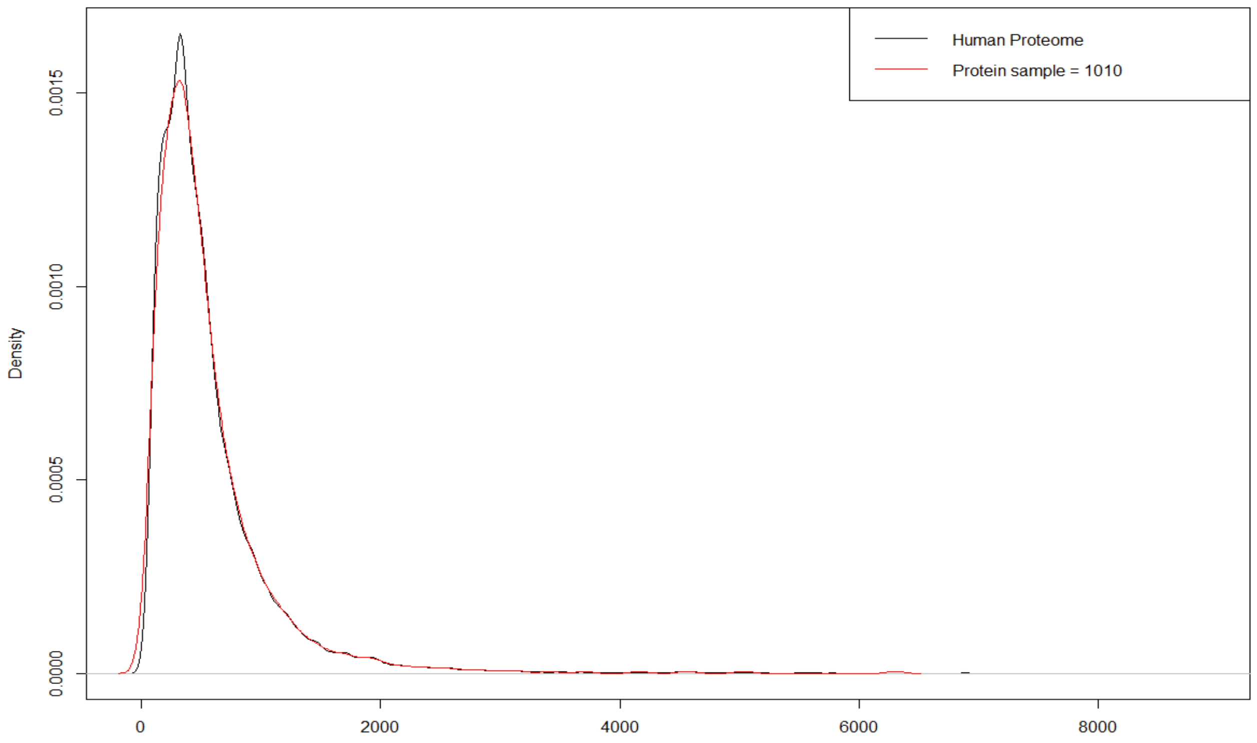


Figure 4.1 Protein length distributions

When multiple lengths are computed in the same run, the execution times are the estimated times presented below multiplied by the number of different lengths. The parameters used for the following benchmarking are specifically:

1. data = HUMAN2016.FASTA
2. nmers= c(9,9)
3. isl = c(1,25), c(1,50), c(1,75), c(1,100)
4. target = “all”
5. filter = TRUE
6. masslist = masslist.txt
7. tolerance = 3
8. as.blocks= TRUE
9. decoy = TRUE
10. ncores = 6

The input file used to test the pipeline was Human proteome 2016 from Uniprot, containing 20191 proteins that were separated in 20 groups of equal size and length distribution. The mass list contained 330.000 elements (peptides). The results are illustrated in **figure 4.2**

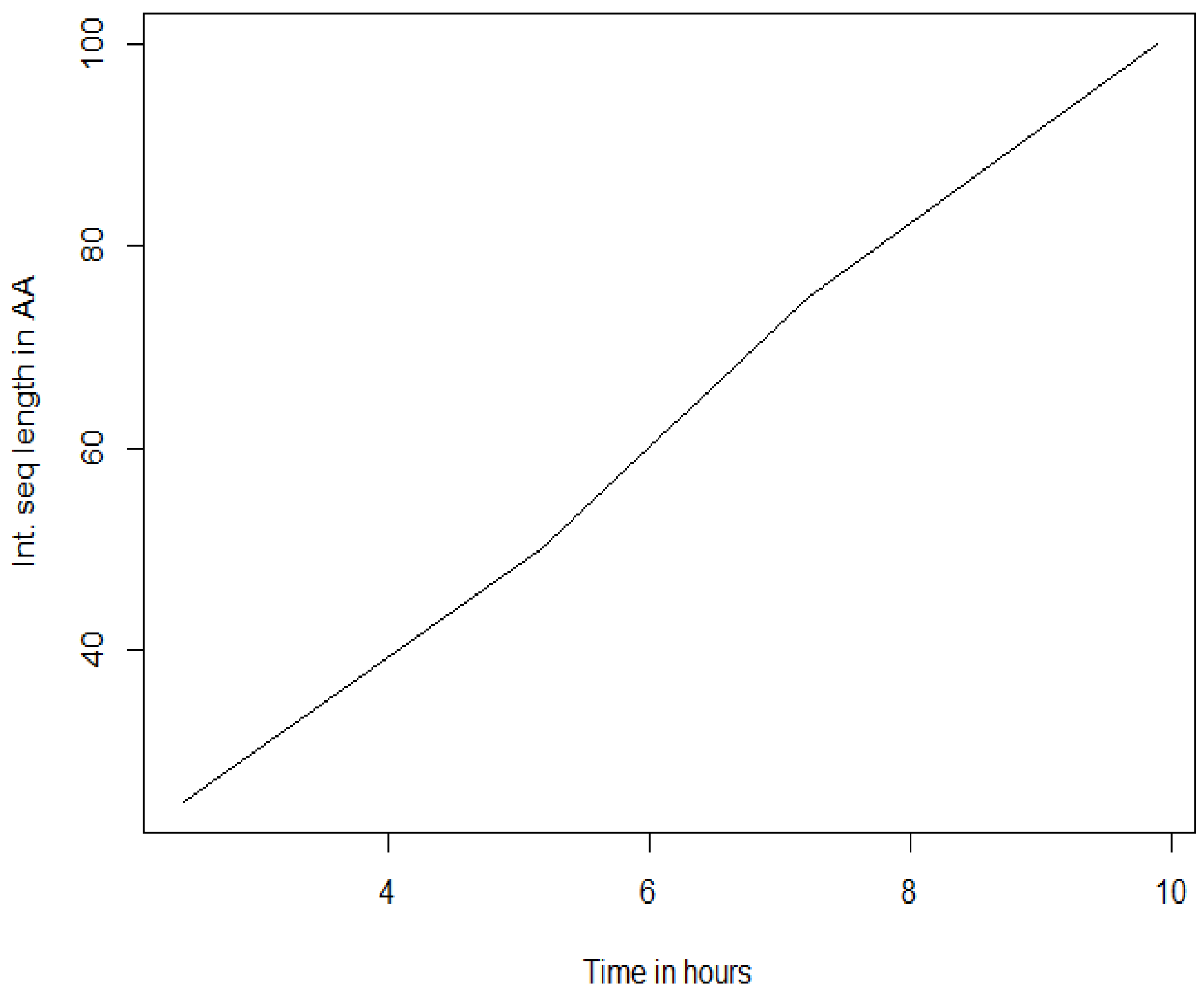


Figure 4.2 : Automated process for 9mers on 1010 proteins using 6 logical cores. Execution time in x axis versus intervening sequence length in y axis.

As illustrated, the longer the intervening sequence length in spliced peptides, the longer the execution time. This is expected as much more fragment combinations are computed in a given sequence for longer intervening AA chains. The relationship of execution time and *isl* looks to be almost perfectly linear. The computational time for non-spliced peptides is always fixed and virtually 0.

Specifically, for intervening sequence length $isl = [1{:}25]$ the elapsed time was $t = 2{,}42\ hours$. For $isl = [1{:}50]$ the elapsed time was $t = 5{,}16$ hours. For $isl = [1{:}75]$, $t = 7{,}82$ hours and for $isl = [1{:}100]$, $t = 10{,}32$ hours. Thus, in a same computer build using the same CPU power, the execution time for the complete human proteome data base will be 20x the times listed.

We also benchmarked the procedure in serial versus parallel mode for the following parameters:

1. data = HUMAN2016.FASTA
2. nmers= c(9,9)
3. isl = c(1,25)
4. target = “all”
5. filter = TRUE
6. masslist = masslist.txt
7. tolerance = 3
8. as.blocks= FALSE
9. decoy = TRUE

In **figure 4.3** we illustrate the gain in speed when using 1, 2, 3, 4, 5 and 6 cores.

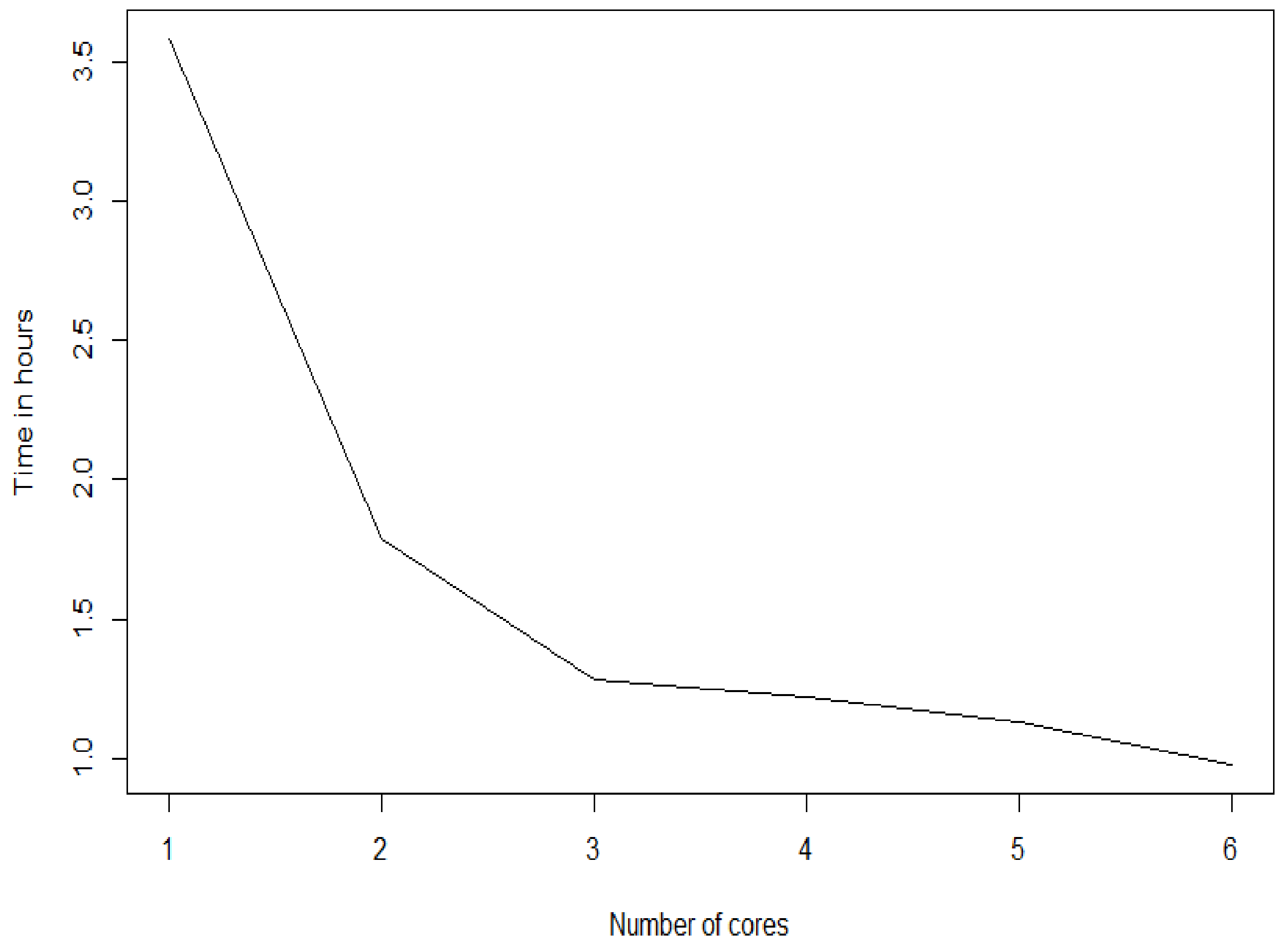


Figure 4.3 Speed gain from parallel execution

4.2.2 Main database construction

Using the same parameters as in 4.2.1 we benchmark the database construction only, for the same group of proteins. The database includes spliced and non-spliced peptides only.

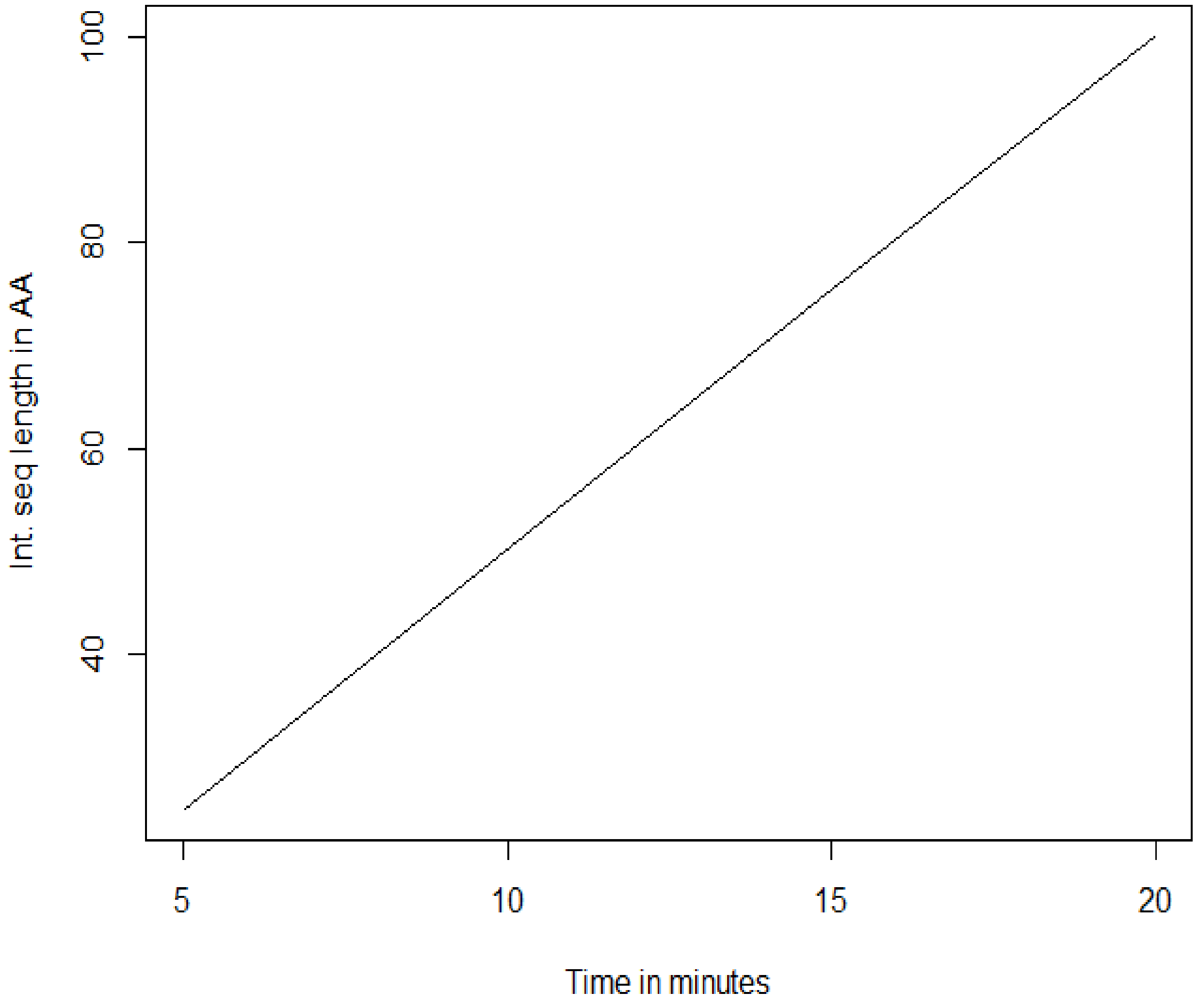


Figure 4.4: Main database construction for 9mers on 1010 proteins and 6 logical cores. Execution time in x axis versus intervening sequence length in y axis.

Similar results as in the previous procedure, however execution time is much shorter here. This is the least computationally intensive procedure.

For intervening sequence length $isl = [1{:}25]$ the elapsed time was $t = 5{,}03$ minutes. For $isl = [1{:}50]$ the elapsed time was $t = 9{,}92$ minutes. For $isl = [1{:}75]$, $t = 14{,}9$ minutes and for $isl = [1{:}100]$, $t = 19{,}96$ minutes.

We also tested the execution time of database computation versus increasing protein lengths. The results are illustrated in **figure 4.5**.

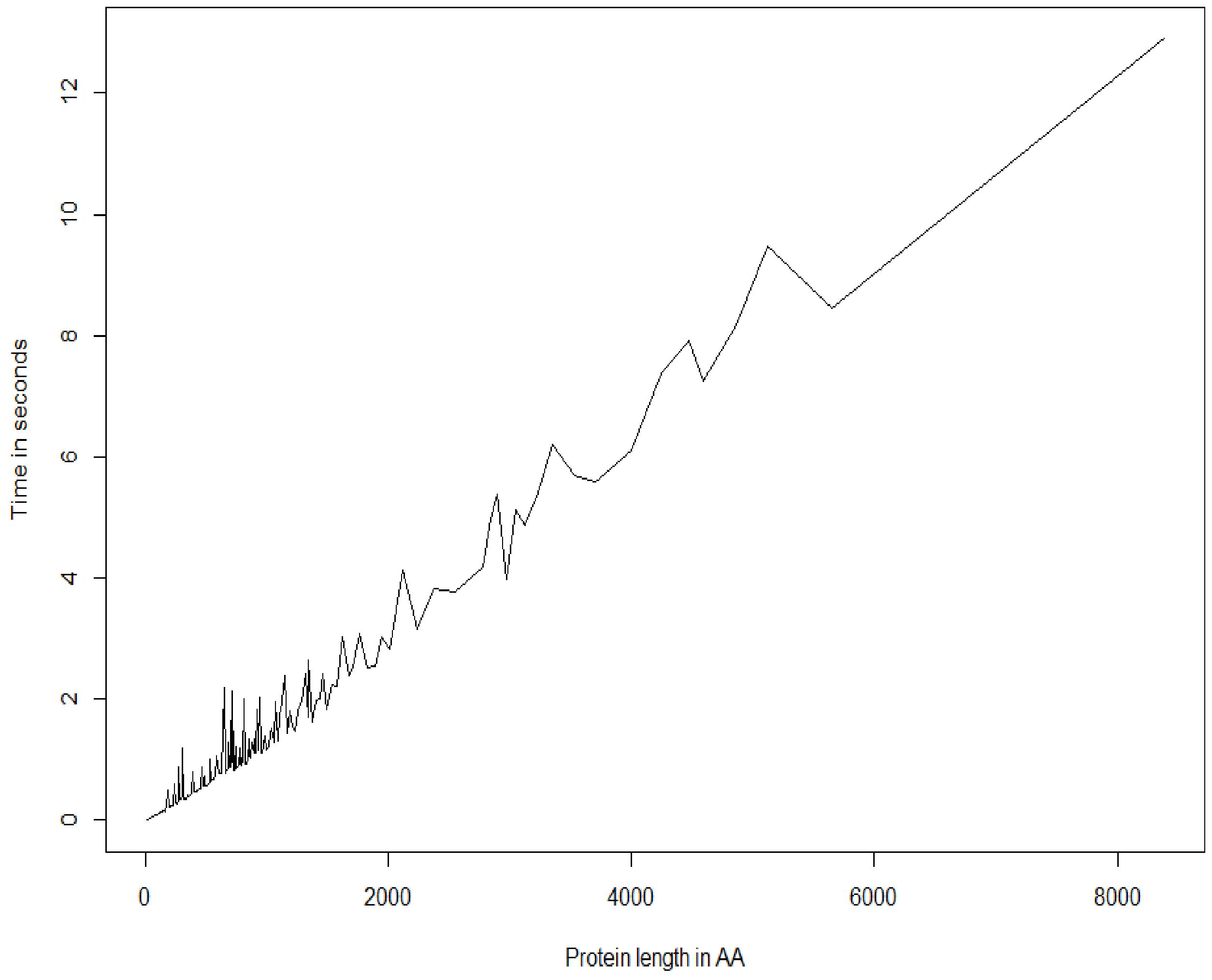


Figure 4.5: Execution time for different protein lengths. Protein length in x axis versus execution time in y axis.

4.2.3 Filtering process

When the filtering is done separately, all Rdata files from the previous procedure are loaded one by one. The filtering was done with 3ppm tolerance using randomly generated mass lists of different lengths. We tested mass lists of length $L = [85000, 170000,\ 260000,\ 350000]$ and no difference in execution time was observed. In **figure 4.6** we illustrate the change in execution time of the filtering process for different protein lengths when running in one cpu core, using a mass list of 350000 rows

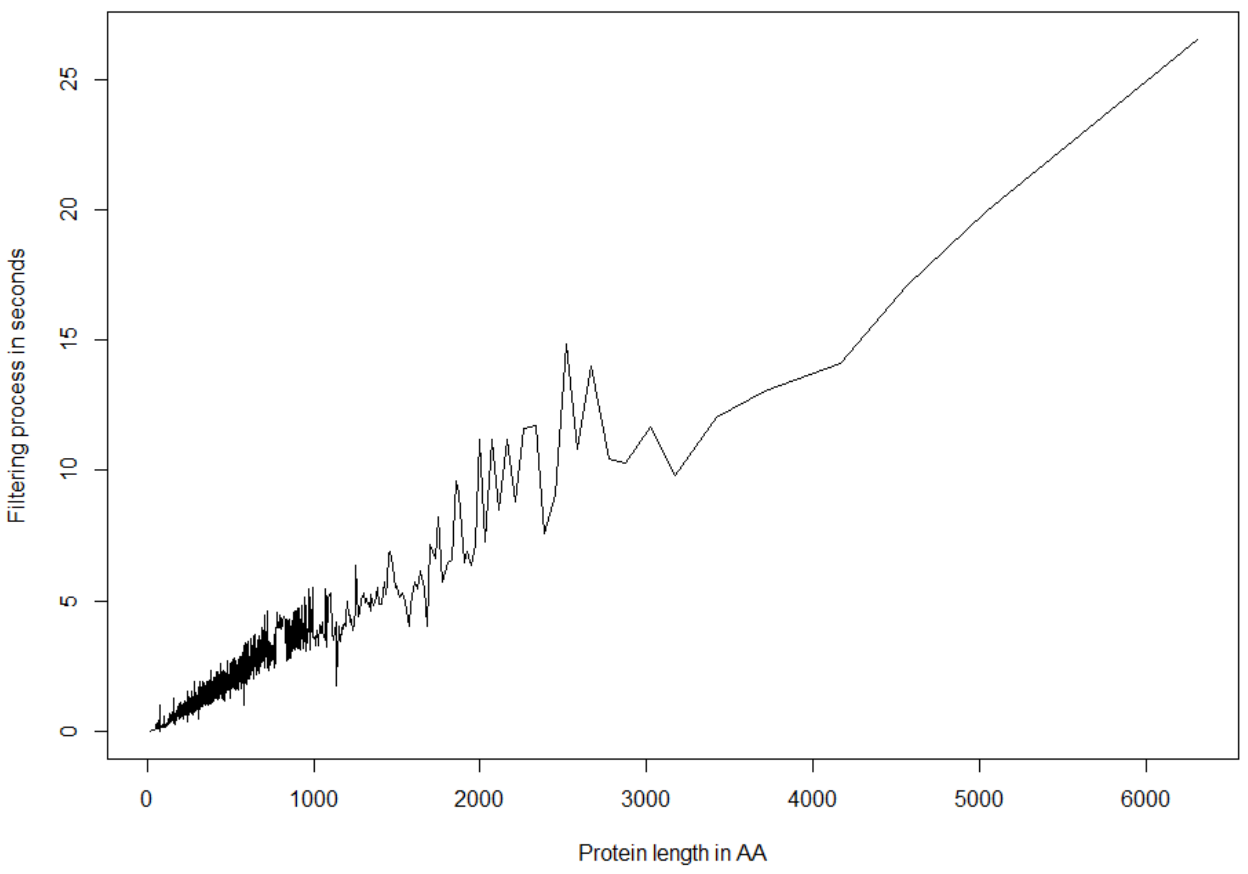


Figure 4.6: Filtering execution time for different protein lengths. Protein length in x axis versus execution time in y axis.

Specifically, single databases of peptides derived from small proteins of up to 1000 amino-acids need between 0.01 to 5 seconds to be filtered. A peptide database derived from a long protein of 5000 amino-acids or more, needs 15 to 20 seconds reaching 60 seconds for unusual large proteins of 20000 or more amino-acids. Total execution time for a group of 1010 proteins, $9mers$ and $isl = [1,25]$ was $t = 9{,}54$ minutes, running on 6 logical cores.

4.2.4 Decoy database construction

Again, when decoys are constructed separately the Rdata files from procedure 4.2.2 are loaded one by one. In **figure 4.7** we illustrate the change in execution time of the decoy construction process for different protein lengths when running in one cpu core, using the reversed method. Once again, linearity is observed between protein length and execution time. Execution time for 1010 proteins, $9mers$ peptides and $isl = [1,25]$ on 6 logical cores was $t = 14{,}2$ minutes.

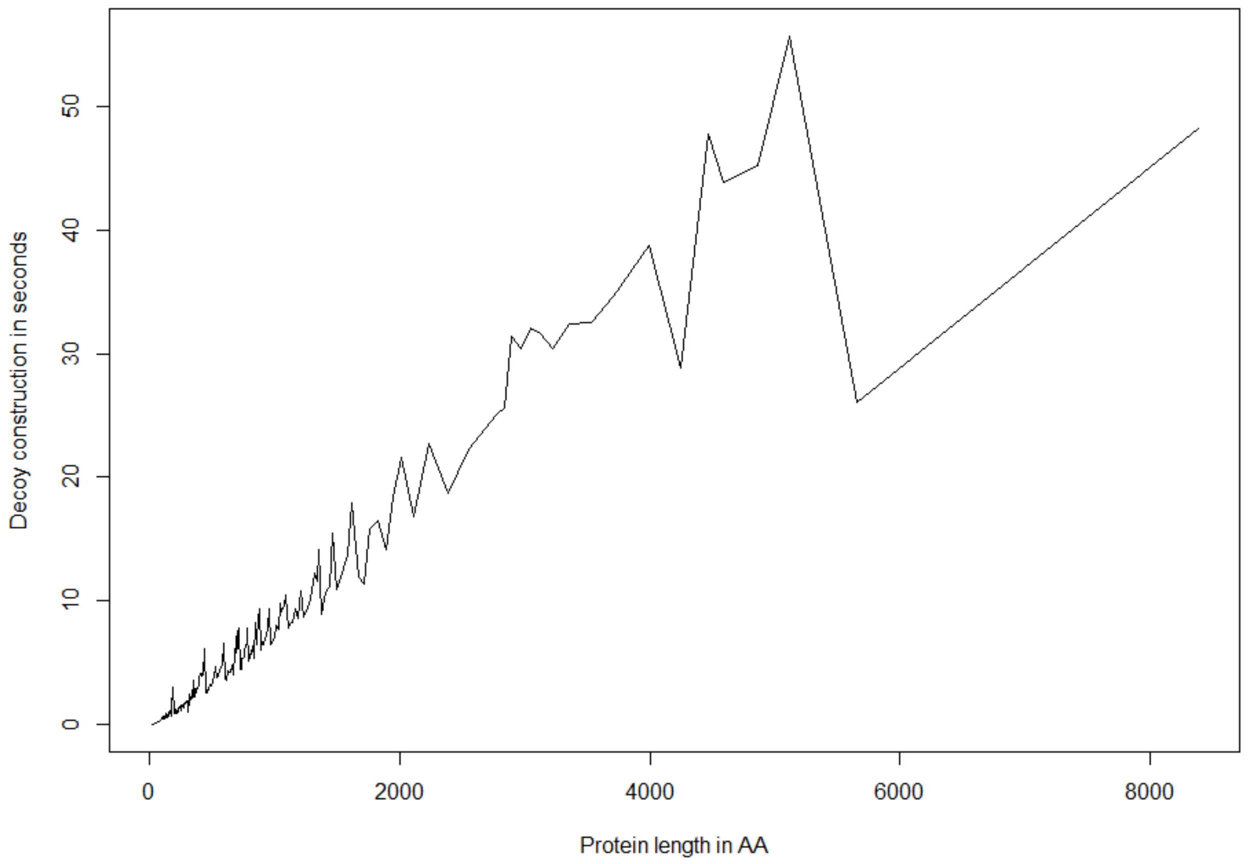


Figure 4.7: Decoy construction for different protein lengths. Protein length in x axis versus execution time in y axis.

4.2.5 Output to blocks

This is the most intensive and time consuming procedure. In **figure 4.8** we illustrate the change in execution time of the format conversion process for different protein lengths when running in one cpu core. The relationship between execution time and protein length looks to be exponential in this process. Due to the low frequency of long proteins i.e. over 4000 amino-acids, in the proteome and, therefore, our sample, the plot line is not smooth after the point $x = 4000$ on x axis. However if more long proteins where present in the sample, the exponential trend would be clearly visible. Total execution time for a group of 1010 proteins, $9mers$ and $isl = [1,25]$ was $t = 120,2$ minutes, running on 6 logical cores.

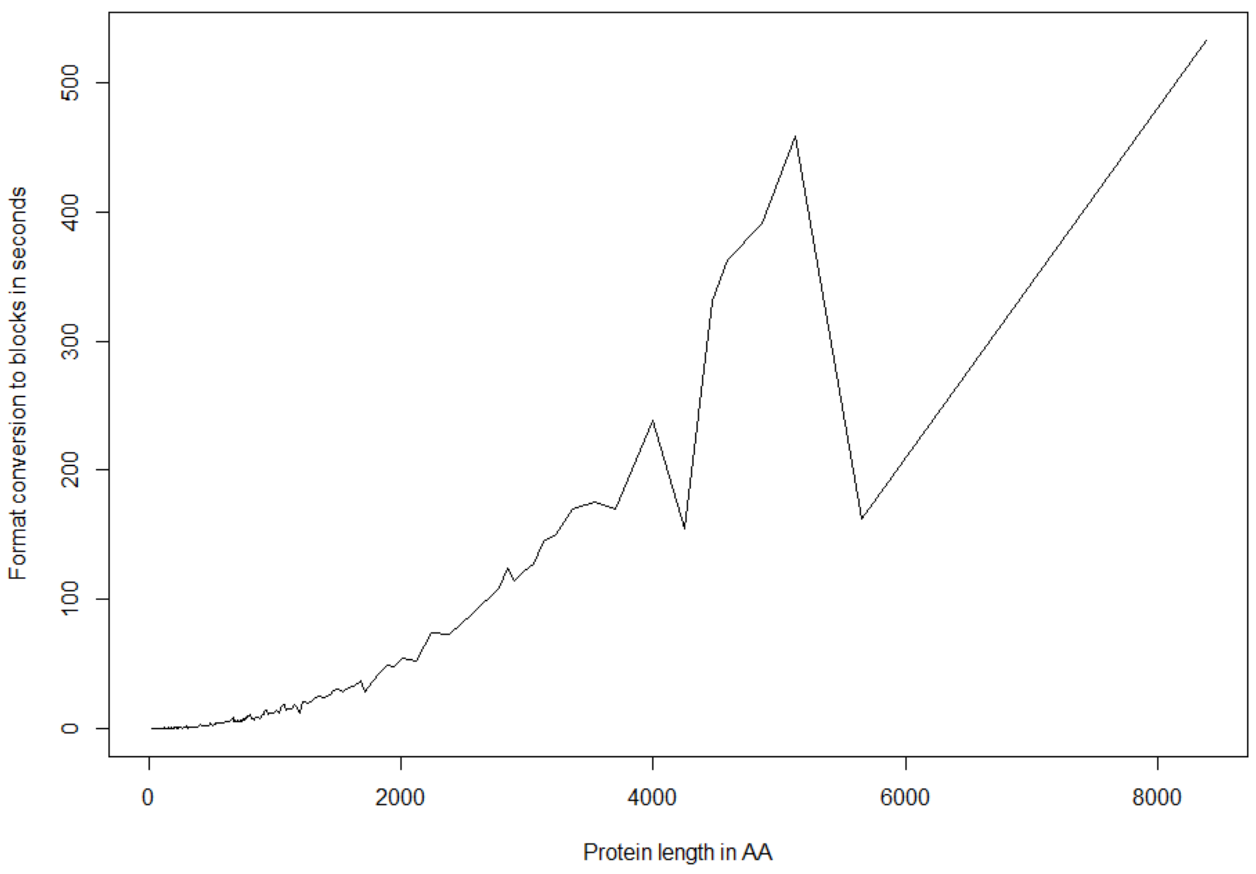


Figure 4.8: Format conversion execution time for different protein lengths. Protein length in x axis versus execution time in y axis.

5. CONCLUSION

The experimental analysis of spliced peptides has garnered significant interest in the recent years. More and more studies prove their definitive contribution to immune system function and the many ramifications of this finding, such as vaccine design, immunotherapy and autoimmune diseases therapy.

To overcome the computational limitations due to lack of an equivalent software solution, we developed ProteoSplicer, a pipeline to initiate the process of spliced-peptides identification. Splicer creates custom spliced and non-spliced peptides databases which are mandatory when using MS/MS analysis software, as well as their decoys. We explained thoroughly in chapter 3 how it works, function by function and benchmarked it in chapter 4. It is an open source pipeline and accessible at Github platform.

Several challenges were faced during the development of the algorithm and the implementation of an A to Z identification analysis. The greatest issue was the execution time when large proteome databases were used or the complete spliced peptide databases were needed. For this we implemented speed optimization techniques in R, and C++ scripts for the most intensive tasks. However, there is still room for improvements as it can get quite slow when large parameters are used. Nonetheless, this issue can be overcome with extra computing power, by either using more cores or more powerful CPU.

Another issue was the unavailability of a simple and specific database search engine software solution. Software for MS analysis is very complex with several features and, usually, not open source. The complexity of their design might arise many different issues for different users, ranging from system compatibility to hardware inadequacy. Spliced peptides databases being so large are even more troublesome to analyze with the open source software available. As the core scoring algorithm that is used to match peptides to experimental spectra is less complex, a far less complex solution could be implemented, like an R package. A future plan of this package is to include a scoring algorithm for matching theoretical peptides to experimental MS/MS spectra that is simple and specific to the spliced-peptides identification process.

ANNEX I

Required packages

The algorithm is dependent on the following R packages:

1. “Seqinr” [25]

2. “Rcpp” [26][27][28]

3. “Foreach” [29]

4. “doSnow” [29]

5. “Peptides” [30]

6. “MASS” [31]

Each of these packages should be installed and loaded before running any script.

All scripts for running the algorithm can be found at:
https://github.com/alevk/ProteoSplicer

REFERENCES

[1] Androlewicz, M. J., Anderson, K. S., and Cresswell, P. (1993) Evidence that transporters associated with antigen processing translocate a major histocompatibility complex class I-binding peptide into the endoplasmic reticulum in an ATP-dependent manner. Proc. Natl. Acad. Sci. U.S.A. 90, 9130-9134

[2] Vigneron N et al.(2017) “Peptide splicing by the proteasome“. The journal of biological Chemistry. 292(51):21170-21179

[3] Hanada, K., Yewdell, J. W., and Yang, J. C. (2004) Immune recognition of a human renal cancer antigen through post-translational protein splicing. Nature 427, 252-256.

[4] Vigneron, N. et al. (2004) An antigenic peptide produced by peptide splicing in the proteasome. Science 304, 587-590

[5] Warren, E. H. et al. (2006*)* An antigen produced by splicing of noncontiguous peptides in the reverse order. Science 313, 1444-1447.

[6] Dalet, A., Vigneron, N. et al.. (2010) Splicing of distant peptide fragments occurs in the proteasome by transpeptidation and produces the spliced antigenic peptide derived from fibroblast growth factor-5. J. Immunol. 184, 3016-3024

[7] Michaux, A et al. (2014) A Spliced Antigenic Peptide Comprising a Single Spliced Amino Acid Is Produced in the Proteasome by Reverse Splicing of a Longer Peptide Fragment followed by Trimming. J. Immunol. 192, 1962-1971

[8] J.Liepe et al. (2016) A large fraction of HLA class I ligands are proteasome-generated spliced peptides

[9] F.Ebstein et al.(2016*)* Proteasomes generate spliced epitopes by two different mechanisms and as efficiently as non-spliced epitopes. Scientific Reports 2016;6: 24032

[10] K.Textoris-Taube et al. (2015) The T210M Substitution in the HLA-a*02:01 gp100 Epitope Strongly Affects Overall Proteasomal Cleavage Site Usage and Antigen Processing.

[11] Mishto M. et al (2012) Driving forces of proteasome-catalyzed peptide splicing in yeast and humans. Moll Cell Proteomics Oct;11(10):1008-23

[12] Webb-Robertson BJ, Cannon WR (2007) Current trends in computational inference from mass spectrometry-based proteomics. Brief Bioinform Sep;8(5):304-17. Epub 2007 Jun 20

[13] Bartels Christian (1990) “Fast algorithm for peptide sequencing by mass spectroscopy” Biological Mass Spectrometry 19(6): 363-368

[14] Sakurai, T.; Matsuo, T.; Matsuda, H.; Katakuse, I. (1984). "PAAS 3: A computer program to determine probable sequence of peptides from mass spectrometric data". Biological Mass Spectrometry. 11 (8): 396–399

[15] Hamm, C. W.; Wilson, W. E.; Harvan, D. J. (1986). "Peptide sequencing program". Bioinformatics. 2 (2): 115–118

[16] Siegel, MM; Bauman, N (1988). "An efficient algorithm for sequencing peptides using fast atom bombardment mass spectral data". Biomedical & Environmental Mass Spectrometry. 15 (6): 333–43

[17] Johnson, RS; Biemann, K (1989). "Computer program (SEQPEP) to aid in the interpretation of high-energy collision tandem mass spectra of peptides". Biomedical & Environmental Mass Spectrometry. 18 (11): 945–57

[18] Scoble et al. (1987). "A graphics display-oriented strategy for the amino acid sequencing of peptides by tandem mass spectrometry". Fresenius' Zeitschrift für Analytische Chemie. 327 (2): 239–245.

[19] Jurgen Cox et al. (2011) “Andromeda: A Peptide Search Engine Integrated into the MaxQuant Environment”

[20] Joshua E.Elias and Steven P.Gygi (2010). Target-Decoy Search Strategy for Mass Spectrometry-Based Proteomics. Methods Mol Biol. 2010; 604: 55–71.

[21] Moore RE, Young MK, Lee TD (2002) . Qscore: an algorithm for evaluating SEQUEST database search results. J Am Soc Mass Spectrom. 2002; 13:378–86.

[22] Peng J, Elias JE, Thoreen CC, Licklider LJ, Gygi SP (2003). Evaluation of multidimensional chromatography coupled with tandem mass spectrometry (LC/LC-MS/MS) for large-scale protein analysis: the yeast proteome. J Proteome Res. 2003;2:43–50

[23] Elias JE, Gygi SP (2007). Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry. Nat Methods. 2007;4:207–14

[24] Kall L, Storey JD, MacCoss MJ, Noble WS (2008). Assigning significance to peptides identified by tandem mass spectrometry using decoy databases. J Proteome Res. 2008;7:29–34.

[25] Charif, D. and Lobry, J.R. (2007)

[26] Dirk Eddelbuettel and Romain Francois (2011). Rcpp: Seamless R and C++ Integration. Journal of Statistical Software, 40(8), 1-18

[27] Eddelbuettel, Dirk (2013) Seamless R and C++ Integration with Rcpp. Springer, New York. ISBN 978-1-4614-6867-7.

[28] Dirk Eddelbuettel and James Joseph Balamuta (2017). Extending R with C++: A Brief Introduction to Rcpp. PeerJ Preprints 5:e3188v1

[29] Microsoft and Steve Weston (2017). foreach: Provides Foreach Looping Construct for R. R package version 1.4.4.

[30] Osorio, D., Rondon-Villarreal, P. & Torres, R. (2015) Peptides: A package for data mining of antimicrobial peptides. The R Journal. 7(1), 4-14

[31] Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0