

**Étudiants**

ABBES Billel

COGGIA Thomas

**Encadrants**

BERNARDES Juliana

ZIADI Tewfik

# **RAPPORT DE PSTL**

## **But4reuse : Adaptateur pour protéines**

# SOMMAIRE

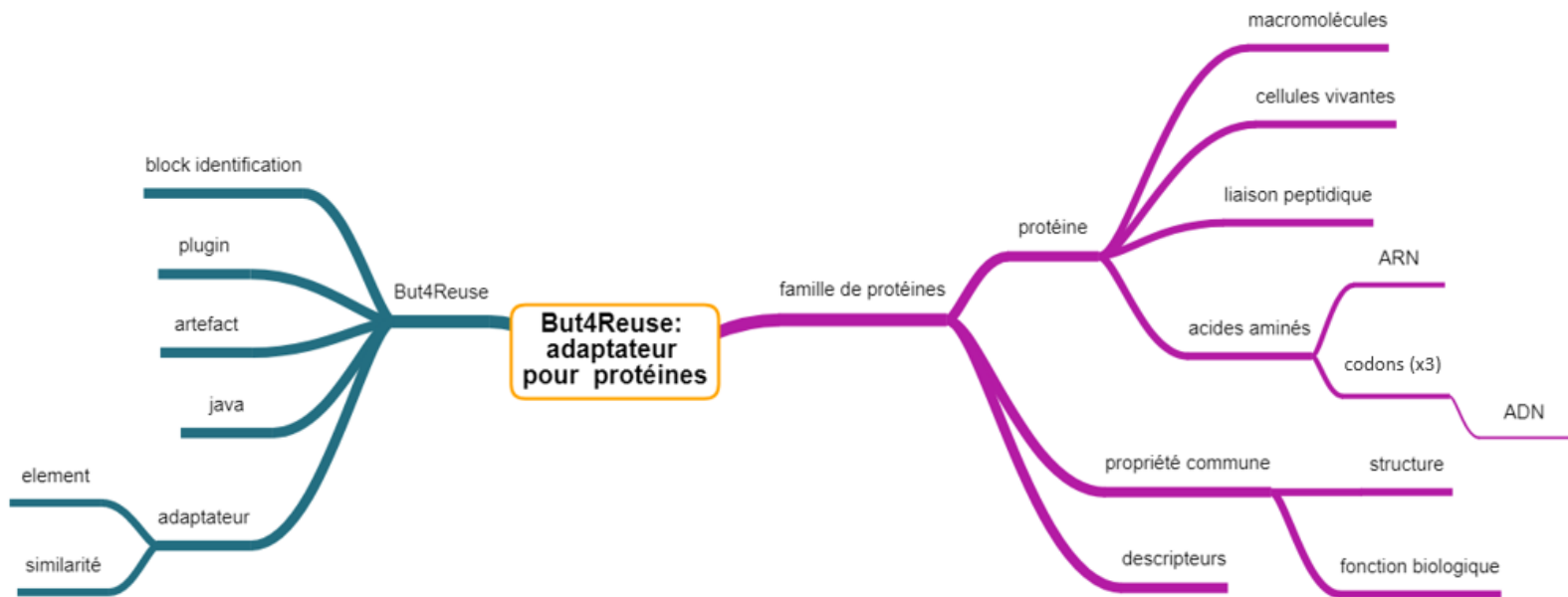
Contexte .....	P3
Pour mieux comprendre les protéines .....	P4
Pour mieux comprendre but4reuse .....	P5
Étapes de l'implémentation de l'adaptateur pour protéines .....	P5
définir de quel type d'élément une protéine est constituée .....	P5
définir à quel point un élément est similaire à un autre .....	P7
définir des mots clés qui définissent des "caractéristiques" de l'élément..	P9
définir, si on le souhaite, le type de dépendance qu'il peut y avoir entre les éléments.....	P9
définir, si on le souhaite, le type de contrainte qu'il peut y avoir sur un élément .....	P9
d'extraire d'un artefact (fichier) une liste d'éléments, avec leur dépendances et leur contraintes .....	P10
Les descripteurs implémentés .....	P11
Sélection des préférences .....	P12
La base de donnée utilisée pour l'étude de séquences ARN.....	P13
Structure de la base de données .....	P13
Remarque sur le format d'une séquence ARN .....	P14
Explication rapide de l'implémentation .....	P15
Première idée développée pour vérifier l'appartenance d'une protéine à une famille de protéines .....	P16
Seconde idée développée pour vérifier l'appartenance d'une protéine à une famille de protéines .....	P19
Diviser un fichier d'une famille de protéines, en plusieurs fichiers de protéines .....	P22
Chronologie de notre travail .....	P23
Manuel d'utilisation But4reuse .....	P24

# Contexte

Les protéines d'une même famille partagent des séquences ayant des caractéristiques proches en terme de structure et de fonction cellulaire. C'est à ce moment qu'intervient But4reuse, logiciel qui permet d'identifier des similarités entre différents objets de même types (aussi appelés artefacts). Ces artefacts peuvent être des images, du code, etc. Le but de notre projet est donc d'utiliser But4reuse pour comparer des protéines d'une même famille.

But4reuse est développé en Java à base de plug-in Eclipse, et offre une structure logicielle générique : But4reuse décompose un artefact en un ensemble d'éléments, et utilise un algorithme, déjà implémenté, qui sait identifier des blocs d'éléments communs qui constituent les différents artefacts étudiés. Mais cet algorithme manipule des interfaces qu'il reste à définir, selon ce que l'on souhaite comparer. Une implémentation de ces interfaces est appelé un adaptateur, d'où l'objectif de notre projet, implémenter un adaptateur pour protéines.

Voici la carte heuristique que nous avons réalisé et transmis dans notre carnet de bord, qui traduit rapidement l'arbre de connaissance de notre projet où l'on distingue une branche But4reuse, et une branche protéine.

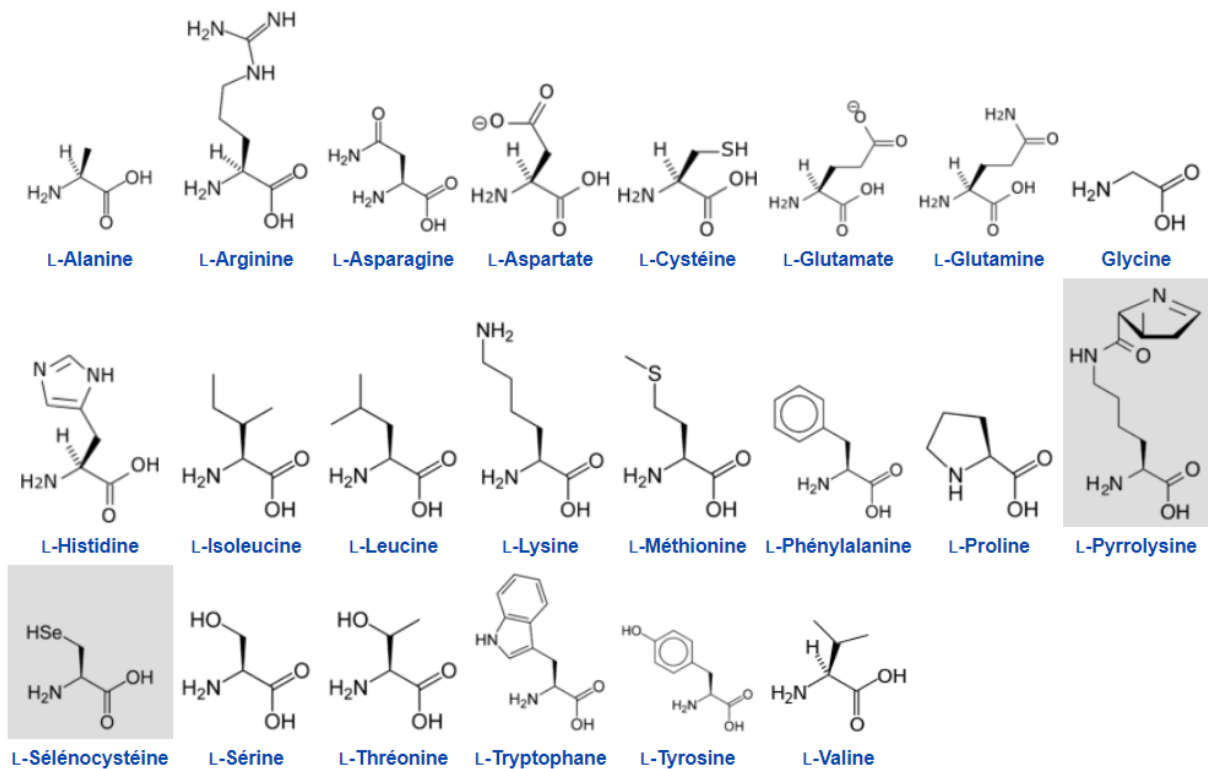


## Pour mieux comprendre les protéines ..

Les protéines sont les principales macromolécules (grosses molécules) qui constituent la vie, avec les glucides et les lipides.

Une protéine n'est rien d'autre qu'une séquence d'acides aminés, et il en existe 20 différents dans les protéines, que l'on représente avec les lettres de l'alphabet. En réalité il en existe 22, mais il s'agit de 2 acides aminés particuliers et rares.

Cette séquence d'acides aminés est ce que l'on appelle une séquence ARN (ou RNA en anglais).



Pour avoir un ordre de grandeur, nous travaillons généralement sur des séquences dont la taille varie entre 50 et 300 acides aminés.

## **Pour mieux comprendre but4reuse ..**

But4reuse va nous permettre dans ce projet de trouver des similarités dans des séquences ARN.

Un impératif de l'adaptateur est de savoir comment décomposer un artefact (un fichier en somme), en un ensemble d'éléments.

Notre travail consiste donc, selon le critère :

- (1) définir de quel type d'élément une protéine est constituée.
- (2) définir à quel point un élément est semblable à un autre, ce que l'on appelle fonction de similarité.
- (3) définir des mots clés qui définissent des "caractéristiques" de l'élément.
- (4) définir, si on le souhaite, le type de dépendance qu'il peut y avoir entre les éléments (à l'image d'un graphe d'éléments).
- (5) définir, si on le souhaite, le type de contrainte qu'il peut y avoir sur un élément.
- (6) extraire d'un artefact (fichier) une liste d'éléments, avec leur dépendances et leur contraintes.

## **Étapes de l'implémentation de l'adaptateur pour protéines**

### **1. définir de quel type d'élément une protéine est constituée.**

La décomposition d'une protéine selon un descripteur nous permet d'avoir un ensemble d'éléments susceptibles d'être analysé pour figurer la présence ou l'absence d'une propriété physiologique ou biochimique.

Les critères de similarité de deux séquences ARN sont très multiples dans le sens où il n'y a pas de réel modèle pour assurer que deux séquences sont plus ou moins similaires : cela dépend surtout de ce que l'on souhaite comparer entre deux séquences.

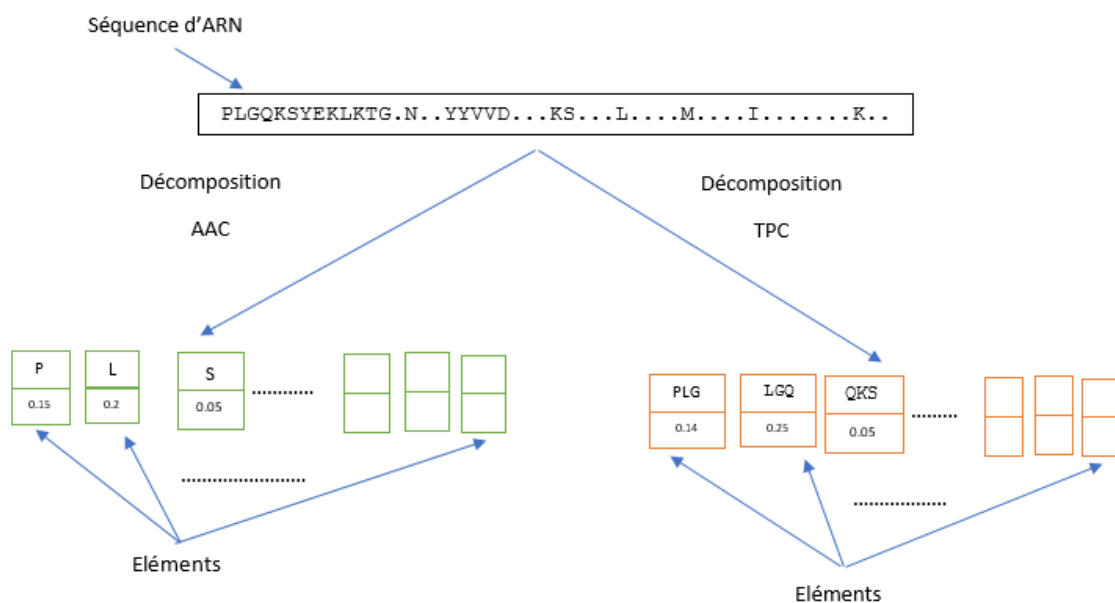
Pour cela, nous nous sommes aidés du projet iFeature qui décrit différents type de données caractéristiques d'une séquence ARN. Ce projet s'est en fait inspiré d'une base de donnée appelée AAindex (Amino Acid Data Base), qui présente de nombreux critères qu'il peut être intéressant d'extraire d'une séquence ARN. Les données extraites selon un critère sont appelées des descripteurs. On parlera alors plus de type de descripteurs, que de critère.

Dans notre projet, les descripteurs jouent le rôle d'éléments qui peuvent être comparées.

Un exemple simple, il s'agit des descripteurs Acid Amino Composition (AAC), qui, pour chaque acide aminé, lui associe sa fréquence dans la séquence. Un descripteur de AAC pourrait donc être le couple (N, 0.07).

Finalement, un type de descripteur permet de définir ce qu'est un élément.

Dans ce projet, nous avons du alors développer un seul adaptateur, mais capable d'analyser des séquences ARN sous des critères différents.



Dans cet objectif, nous avons eu l'idée de définir une base commune aux éléments de descripteurs (ce qui correspond à une classe abstraite *IDescriptorElement*), qui permet de savoir dans quel cadre l'élément évolue (entre-autres), à savoir :

- fournir la séquence depuis laquelle l'élément a été calculé
- fournir le nom du type de descripteur auquel il correspond

De plus, parmi les types de descripteurs présentés par *iFeature*, il est possible de dégager des points communs parmi certains d'entre eux. Cela permet de factoriser le projet, mais aussi de rendre le projet favorable à l'ajout de nouveaux descripteurs, ainsi que de fournir des descripteurs plus génériques que ceux proposés. Dans ces familles de types de descripteurs, on retrouve notamment :

- Famille qui associe une donnée avec une fréquence  
(ex : AAC associe un acide aminé (une lettre), avec une fréquence)  
(cf. *SubSequenceFrequencyDescriptorElement*)

- Famille dont les descripteurs sont composés eux-mêmes de descripteurs (ex : EAAC est une analyse AAC sur toutes les  $n - k$  sous-séquences) (cf. *SubSequenceDescriptorElement*)
- Famille utilisant la notion de groupe (à savoir associer un identifiant à un ensemble d'acides aminés, et donc grouper l'analyse). (ex : GAAC, EGAAC, GDPC, GTPC, CTDC, CTDI, etc)

Nous présentons un peu plus loin dans le rapport les types de descripteurs que nous avons implémentés.

## 2. définir à quel point un élément est similaire à un autre

Une fois le type de descripteur (critère) choisi, il faut pouvoir évaluer la similarité entre deux éléments et ainsi fournir une valeur entre 0 et 1, cela dépend tout d'abord du critère étudié.

Ainsi, pour chaque type de descripteur, il faut choisir une heuristique de comparaison. Celle-ci est soit fixée dans le code du projet (par choix), ou paramétrable en runtime (via la page de préférences).

En effet, lorsqu'il s'agit de comparer des chaînes de caractères, très généralement, la similarité a un comportement binaire (chaînes égales, ou non) ; le problème est plus complexe dans le cas de valeurs arithmétiques.

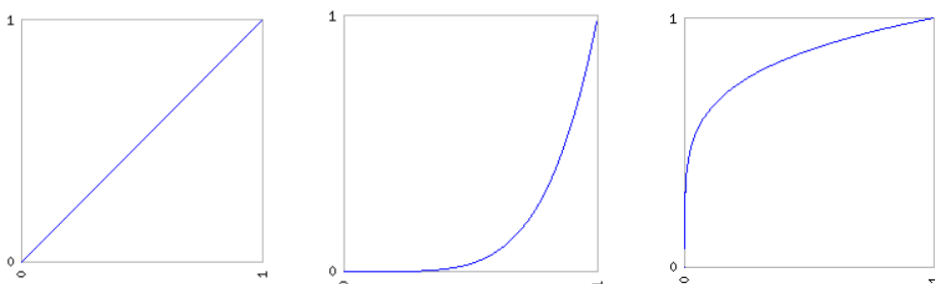
Lorsque l'on compare la similarité entre deux valeurs arithmétiques, on cherche (généralement, car cela pourrait être un autre critère) à connaître à quel point ces valeurs sont proches.

Pour cela on calcule la différence des deux valeurs, et ce en valeur absolue, puisque l'ordre dans la comparaison de deux éléments n'a pas à intervenir dans l'algorithme d'identification des blocs.

Ensuite, la différence de cette valeur doit être travaillée, pour renvoyer une valeur entre 0 et 1. Nous distinguons deux types de données arithmétiques à comparer dans les descripteurs que nous avons implémentés :

- **Des valeurs réelles entre 0 et 1 (comme les fréquences)**

La différence est donc comprise elle aussi entre 0 et 1, et les fonctions mathématiques "usuelles" ne manquent pas non plus :



- La plus simple est la fonction linéaire  $f(x) = x$ .
- On peut aussi utiliser  $f(x) = x^n$ , qui croît doucement, et donne alors réellement de l'importance qu'aux valeurs très proches, le phénomène étant amplifié quand  $n$  est grand (avec  $n = 1$ , on retrouve la fonction linéaire précédente).
- $f(x) = x^{(1/n)}$  qui a l'effet inverse : peut-être moins intéressante, elle donne un peu plus de crédit aux valeurs un peu éloignées.
- Et on peut s'inventer des fonctions un peu moins usuelles, comme  $f(x) = 0$  pour les valeurs de  $x$  comprises entre 0 et 0.9, puis linéaire à partir entre 0.9 et 1. Il ne reste qu'à les définir, et les proposer dans les préférences.

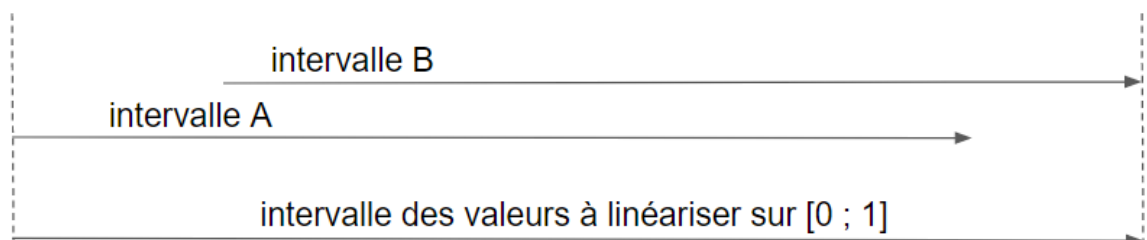
Bien sûr, on prend  $x = (1 - \text{diff})$  dans ces exemples.

- **Des valeurs réelles bornées**

Dans ce cas là, il suffit de linéariser les valeurs comprises entre  $[\min ; \max]$  sur  $[0 ; 1]$ .

- **Des valeurs réelles, bornées selon la séquence**

Le problème est ici un peu plus subtil. Une fausse bonne idée est de prendre le min des deux minimums, et le max des deux maximums, provenant des deux séquences, et ainsi on peut linéariser à l'image des valeurs réelles bornées.



Cependant cette solution peut amener à avoir des séquences plus “éloignées” en terme de similarité, à cause d'un petit intervalle, et inversement des séquences plus proches grâce à un intervalle plus large (et donc une “compression forte” des valeurs).

Ainsi il est préférable d'opter pour une autre heuristique, par exemple la fonction du seuil, qui considère égal (renvoie 1) lorsque la différence est inférieure au seuil, et sinon renvoie un résultat  $\text{seuil} / \text{diff}$  (plus la différence est grande, plus la valeur temps vers 0, linéairement).

On peut aussi rendre l'heuristique encore plus générique, avec des comportements différents (à savoir non linéaire, contrairement à  $\text{seuil} / \text{diff}$ ), ce qui peut, selon la répartition des valeurs, être plus fin.



- **Nous n'avons pas eu le cas de valeurs entières**, mais s'il devait se présenter, on aurait pu imaginer la fonction d'égalité (résultat binaire, 0 ou 1), qui n'était pas adapté jusque là avec les valeurs réelles.

### **3. définir des mots clés qui définissent des “caractéristiques” de l'élément.**

Dans But4reuse, il est nécessaire de décrire un élément par une chaîne de caractère, pour que celle-ci soit ensuite découpée en un ensemble de mots clés. En effet, ce mécanisme permet, entre-autres, d'avoir un visuel assez rapide sur la composition d'un bloc, pour ainsi dire la prédominance d'un mot clé dans un bloc d'éléments, alors affiché plus gros dans un nuage de mots dans l'interface de But4reuse.

Dans le cadre de notre projet, le texte qui définit un élément dépend bien évidemment du type de descripteurs étudié, mais le parsing qui permet au logiciel de déduire les différents mots clés qui constituent la chaîne, a été réétudié et utilisé dans tous les éléments de descripteur (de sorte à ne plus considérer certains caractères spéciaux comme séparateurs, contrairement à l'implémentation par défaut).

### **4. définir, si on le souhaite, le type de dépendance qu'il peut y avoir entre les éléments.**

Les descripteurs que nous avons utilisés dans le projet sont surtout statistiques, et ne présentent pas vraiment de dépendances qui permettent de diriger l'identification des blocs.

### **5. définir, si on le souhaite, le type de contrainte qu'il peut y avoir sur un élément.**

Les contraintes sont surtout utiles pour que le logiciel puisse reconstruire une protéine à partir des blocs identifiés. Dans le cadre de notre projet, on souhaite surtout s'intéresser aux similarités d'une famille, et surtout le bloc commun à toutes les protéines étudiées. Les contraintes entre blocs, n'était donc pas vraiment une priorité dans notre développement.

## **6. d'extraire d'un artefact (fichier) une liste d'éléments, avec leur dépendances et leur contraintes..**

Cette étape se décompose en deux temps.

Le logiciel But4reuse prévoit une méthode à implémenter qui, à partir d'un fichier, doit renvoyer une liste d'éléments.

Nous avons alors décidé de préciser et donc de détacher l'étape d'analyse d'une séquence ARN, de la lecture même du fichier avec la gestion d'exceptions (bon format de fichier, etc), la gestion des préférences (choix du type de descripteurs utilisé), et le parsing d'une séquence ARN (qui a un format particulier dont nous parlerons un peu plus loin).

L'analyse consiste à produire des éléments à partir d'une chaîne de caractère (séquence ARN). Les éléments produits sont alors, dans la partie haute de l'implémentation, ajoutés dans une liste et celle-ci est renvoyée par la méthode, comme le prévoit la spécification du logiciel.

Finalement, l'implémentation d'un type de descripteurs se résume en deux classes : l'élément qui définit le type de descripteur, et la classe analyseur qui s'occupe de lire une séquence ARN, et produire ces éléments.

Parmi les analyseurs, on retrouve aussi des familles d'analyseurs, à l'image des familles de descripteurs :

- Analyseur qui extrait une "propriété" (une certaine donnée) de chaque sous-séquence de taille k, et produit un élément qui associe, pour chaque type de propriété, une fréquence.
  - ex : DPC, TPC, consiste à connaître la fréquence de paire / triplet, la propriété est la chaîne en elle-même.
  - ex : GDPC, GTPC, consiste à connaître la fréquence de paire / triplet en terme de groupes, la propriété est une liste d'identifiants de groupes.
- Analyseur qui produit des descripteurs de descripteurs, en appelant un sous-analyseur sur des sous-séquences de taille k
  - ex : EAAC consiste à lancer une analyse AAC sur des sous-séquences de taille k.

# Les descripteurs implémentés

## **AAC**

Associe à un acide aminé sa fréquence

## **EAAC**

Pour chaque sous-séquence continue, associe à un acide aminé, sa fréquence "locale".

## **CKSAAP**

Associe à chaque paire d'acide aminés séparés par k espaces, sa fréquence. Il s'agit ici d'un descripteur paramétré.

## **DDE**

Réalise une étude statistique (à partir de variances, etc) sur des paires d'acides aminés : les résultats de cette étude correspondent à des réelles, sans bornes supérieures.

## **DPC, TPC**

Associe à chaque paire / triplet, sa fréquence.

## **GAAC, EGAAC, CKSAAGP, GDPC, GTPC**

Ces types de descripteurs sont basés sur le même principe que ceux présentés au dessus, la différence s'appuie sur le fait que l'étude repose sur un ensemble d'acides aminés, et non plus à un acide aminé. Ces types de descripteurs sont donc paramétrés par les k ensembles disjoints d'acides aminés (dans iFeatures, le groupement des acides aminés est fixé par 5 ensembles).

## **CTDC**

Ce type de descripteurs fonctionne comme un les descripteurs groupé, et est en fait équivalent à GAAC. Il est en fait paramétrable par des groupements de taille 3 (groupes CTD), qui ont un sens biologique (par exemple la caractéristique d'hydrophobicité, 3 ensembles d'acides aminés polaire / neutre / hydrophobe, ou encore la charge électrique, positive / neutre / négative).

Notre implémentation de GAAC étant une version générique, nous proposons dans le code un accès direct aux différents groupes CTD.

## **CTDT**

S'appuie aussi sur les groupes CTD, sauf qu'il s'agit d'évaluer la fréquences des paires en tenant compte de la commutativité.

Par exemple, la fréquence de, positif / neutre et neutre / positif. S'intéresser aux, positif / négatif et négatif / positif, n'a pas de sens (pas de transitions entre ces deux états sur l'aspect biologique).

## Sélection des préférences

Les préférences permettent, dans le cadre de notre projet, de choisir différents critères :

- Le type de descripteurs à utiliser
- Pour certains types de descripteurs, le comportement de la fonction de similarité (linéaire, quadratique, etc) avec de possibles paramètres (comme la valeur d'un seuil)
- Pour certains types de descripteurs, des valeurs qui les paramétrisent (taille d'une sous-séquence, d'un espacement, etc)

De plus, le logiciel But4reuse propose de nombreux outils pour paramétrer l'utilisation que l'on peut en faire. En ce qui concerne l'identification des blocs, l'outil le plus intéressant est celui qui permet de définir à partir de quelle valeur deux éléments sont considérés similaires. Par défaut, cette valeur est fixée à 1.

Cependant, il est judicieux de réduire cette valeur selon le besoin et selon le type de descripteurs utilisé : en effet, lorsque les descripteurs sont définis par des fréquences, ou plus généralement, par des valeurs réelles où l'égalité ne prend pas de sens, l'intervalle fini dans les réels  $[1 ; 1]$  n'est pas adapté. Un intervalle non discret, et donc infini dans les réels, tel que  $[0.97 ; 1]$  est fortement conseillé.

Par exemple, si l'on réalise une analyse AAC sur deux séquences, la première de taille 80, et la seconde de taille 81, et que chacune des deux séquences présentent  $n$  acides aminés D, les deux fréquences de l'acide aminé D seront proches mais pas identiques ( $n / 80$  et  $n / 81$ ). Avec une marge de 3% (soit l'intervalle  $[0.97 ; 1]$ ), il y aura bien similarité.

De même, il est possible d'avoir des types de descripteurs qui présentent des valeurs relativement faibles à cause de leur fonction de similarités, et il serait peut-être justifier de fixer un intervalle plus large,  $[0.9 ; 1]$  par exemple. Il revient cependant au développeur du descripteur de définir une fonction de similarité qui renvoie des valeurs assez bien réparties sur  $[0 ; 1]$ .

# La base de données utilisée pour l'étude de séquences ARN

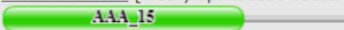
Pour pouvoir étudier des séquences de protéines (selon leur appartenance à une famille), nous avons utilisé la base de données Pfam, qui nous a d'ailleurs été très utile pour comprendre comment sont classifiées les séquences de protéines dans la base de données.

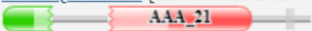
Browse Pfam families

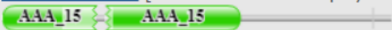
New	Top twenty	Numbers	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
-----	------------	---------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---


ID	Accession	Type	Number of sequences		Average length	Average %id	Average coverage	Has 3D	Change status	Description
			Seed	Full						
A-2_8-polySI	PF07388	Family	3	14	245.60	16	76.25	✓	Changed	Alpha-2,8-polysialyltransferase (POLYST)
A1_Propeptide	PF07966	Motif	133	939	27.90	38	7.32	✓	Changed	A1 Propeptide
A2L_zn_ribbon	PF08792	Domain	12	81	31.60	31	13.49		Changed	A2L zinc ribbon domain
A2M	PF00207	Family	78	5499	89.80	23	5.59	✓	Changed	Alpha-2-macroglobulin family
A2M_BRD	PF07703	Domain	632	4765	145.20	18	9.23	✓	Changed	Alpha-2-macroglobulin bait region domain
A2M_recept	PF07677	Domain	269	2376	87.90	29	6.66	✓	Changed	A-macroglobulin receptor binding domain
AAA	PF00004	Domain	207	131444	130.30	25	21.50	✓	Changed	ATPase family associated with various cellular activities (AAA)
AAA-ATPase_like	PF09820	Family	59	3082	245.30	26	53.86		Changed	Predicted AAA-ATPase
AAA_10	PF12846	Domain	11	2126	302.90	19	41.84		Changed	AAA-like domain
AAA_11	PF13086	Domain	45	18966	184.70	17	19.69	✓	Changed	AAA domain
AAA_12	PF13087	Domain	107	16968	192.80	25	16.21	✓	Changed	AAA domain
AAA_13	PF13166	Domain	16	965	469.00	14	73.39		Changed	AAA domain
AAA_14	PF13173	Family	220	7548	126.40	24	31.17		Changed	AAA domain
AAA_15	PF13175	Domain	38	5862	241.20	12	51.59		Changed	AAA ATPase domain
AAA_16	PF13191	Domain	135	16763	173.30	18	16.05	✓	Changed	AAA ATPase domain

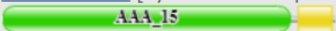
En réalité, une protéine est un assemblage (architecture) de sous-séquence ARN spécifiques, qui ont des particularités (biologique, structurelle). Les sous-séquences qui ont les mêmes particularités et qui ont été retrouvé dans différentes protéines séquencées, sont alors classifiées en familles ou domaines.


There are 3458 sequences with the following architecture: **AAA\_15**  
[Y0KJX4\\_9PROT](#) [Methylophilaceae bacterium 11] Uncharacterized protein {ECO:0000313|EMBL:EUJ11421.1} (484 residues)  
  
[Show](#) all sequences with this architecture.

There are 814 sequences with the following architecture: **AAA\_15, AAA\_21**  
[W4LWQ0\\_9BACT](#) [Candidatus Entothionella factor] Uncharacterized protein {ECO:0000313|EMBL:ETX02196.1} (433 residues)  
  
[Show](#) all sequences with this architecture.

There are 473 sequences with the following architecture: **AAA\_15 x 2**  
[Z5XSL2\\_9GAMM](#) [Pseudoalteromonas lipolytica SCSIO 04301] Endonuclease {ECO:0000313|EMBL:EWH06175.1} (548 residues)  
  
[Show](#) all sequences with this architecture.

There are 197 sequences with the following architecture: **AAA\_23, AAA\_15**  
[W4AGQ3\\_9BACL](#) [Paenibacillus sp. FSL R5-192] Uncharacterized protein {ECO:0000313|EMBL:ETT30663.1} (384 residues)  
  
[Show](#) all sequences with this architecture.

There are 138 sequences with the following architecture: **AAA\_15, DUF3696**  
[K9Z732\\_CYAAP](#) [Cyanobacterium aponinum (strain PCC 10605)] Uncharacterized protein {ECO:0000313|EMBL:AFZ54954.1} (469 residues)  
  
[Show](#) all sequences with this architecture.

There are 80 sequences with the following architecture: **AAA\_15 x 2, DUF3696**  
[V4J8F0\\_9DELT](#) [Uncultured Desulfobacter sp. PB-SRB1] Uncharacterized protein {ECO:0000313|EMBL:ESQ08590.1} (472 residues)  
  
[Show](#) all sequences with this architecture.

Un fichier de la base de donnée se décompose alors en une liste de sous-séquences de protéines : chaque ligne correspond à une sous-séquence précédée de l'identifiant de la protéine où elle a été trouvée, et de l'intervalle où se situe la sous-séquence.

```

714 G4VEV6_SCHMA/15-44 .....VV..RIPLHPLKSAQRTLIEfE...TSLEIVKK--vw.....
715 A0A287B096_PIG/33-51 .....--..RIFLKKMPVRESLKE.R...G-----v.....
716 Q5BKJ1_XENTR/17-45 .....LI..RVPLHKSKTIRETMKE.K...GVLKEFMKTH.....
717 H0XLY6_OTOGA/20-45 .....--..-VPLMKVKSMRENQe.N...GMLKEYLEKY.....
718 H0Z9B1_TAEGU/1-27 .....--..RIPLIRFKSIKKQLKE.K...GELEEFWRNH.....
719 A0A0N4UV42_ENTVE/22-47 .....v-L..RIPLKKQKTIREQLVE.T...NSWHEY----v.....
720 A0A091KUD2_9GRUI/1-24 .....--..RIALRRMPSIRQTLREmG...VKVSD-----v.....
721 A0A226PYA9_COLVI/22-50 .....lk--..RVTLTRHRSRLRKSRA.R...GQLSQFWKAH.....
722 A0A151NYJ2_ALLMI/16-45 .....lv-T..KVPLKKGKTMQRNLKE.H...GKLQDYLKKH.....
723 H3B4C1_LATCH/20-46 .....II..RIPLRKFRMTMRRTMSDgR...MSIEE-----lk.....
724 H2ND88_PONAB/17-45 .....MY..KVPLIRKKSLRRTLSE.H...GLLKDFLKKH.....
725 E1BI12_BOVIN/12-44 .....yim--..KIPLRRVKTMRKTLTE.KkkkNILNNFLKKH.....
726 F6YAL3_CALJA/21-50 .....LV..RIPLHKFTSIRRTMSEmG...GPVEDLIAK-g.....

```

### Remarque sur le format d'une séquence ARN

Ces familles ont une taille “théoriquement” fixée, mais dans la pratique, des morceaux de séquences peuvent parfois manquer, voire se retrouve morcelées à différents endroits dans la séquence ARN de la protéine : en effet, en 3D, la protéine peut se contracter et ainsi créer des “ponts” dans la séquence ARN, ce qui fait alors apparaître les particularités biologiques de la fameuse sous-séquence.

Ainsi, les sous-séquences du même domaine (ou famille), sont alignés à l'aide de caractères “-” lorsque la séquence est “plus courte qu'attendue”, et donc qu'un acide aminé est comme manquant, et les séquences sont aussi agrémentés de lettres minuscules pour présenter les acides aminés qui sont “sous un pont”, et qui n'appartiennent donc pas à la séquence du domaine. Des “.” sont alors ajoutés aux autres séquences pour les aligner avec ces minuscules.

Dans la base de données, on peut alors sélectionner différents formats, (avec ou sans “-”, “.”, minuscules), mais aussi choisir de télécharger seulement les sous-séquences les plus “représentatives” du domaine (seeds).

	Seed (133)	Full (939)	Representative proteomes				UniProt (1631)	NCBI (2776)	Meta (0)
			RP15 (112)	RP35 (283)	RP55 (686)	RP75 (851)			
Alignment:	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Format:	Selex ▼								
Order:	<input checked="" type="radio"/> Tree		<input type="radio"/> Alphabetical						
Sequence:	<input checked="" type="radio"/> Inserts lower case		<input type="radio"/> All upper case						
Gaps:	Gaps as “.” or “-” (mixed) ▼								
Download/view:	<input checked="" type="radio"/> Download		<input type="radio"/> View						
Generate									

Dans le cadre de notre projet, nous ne nous intéressons pas aux alignements, et ignorons les ponts : ces variantes n'ont en faites que très peu d'intérêt puisque nos descripteurs sont essentiellement statistiques. Cependant, nous avons tout de même implémenté un parsing qui permet de filtrer, ou non, la séquence (avec ou sans “-”, “.”, minuscules).

## Explication rapide de l'implémentation

Pour décomposer les séquences d'ARN en éléments on a dû créer une classe ***ProteinAdapter*** qui implémente l'interface *IAdapter*, cette dernière possède une fonction qui rend une ***List<IElement>*** en lui passant le fichier qui contient la protéine à décomposer (ainsi que le moteur de progression, de sorte à avoir un retour graphique sur l'avancé de l'analyse). En implémentant cette fonction on peut définir la manière dont on peut décomposer notre fichier.

L'interface ***IElement*** propose directement une méthode ***similarity*** qui prend en paramètre un autre ***IElement*** pour le comparer à ***this*** et ainsi renvoyer une valeur sur [0 ; 1]. Généralement, un élément consiste à stocker les données qui feront l'objet de l'évaluation de la similarité.

Afin de définir les préférences nous avons créé une première classe ***PreferenceInitializer*** qui consiste à initialiser les valeurs par défaut des préférences. Nous avons également créé une deuxième classe ***ProteinAdapterPreferencePage***, où sont définis les différents composants de sélections que l'on retrouve dans la fenêtre des préférences.

De plus, nous avons un package descriptors, qui classifie les différents descripteurs implémentés. Ainsi, pour chaque descripteurs implémentés, un package lui est attribué au sein de ce package descriptors, contenant un analyseur et l'élément du descripteur.

Un package heuristic contient les différentes fonctions de comparaisons dont nous avons besoin pour comparer des valeurs arithmétiques.

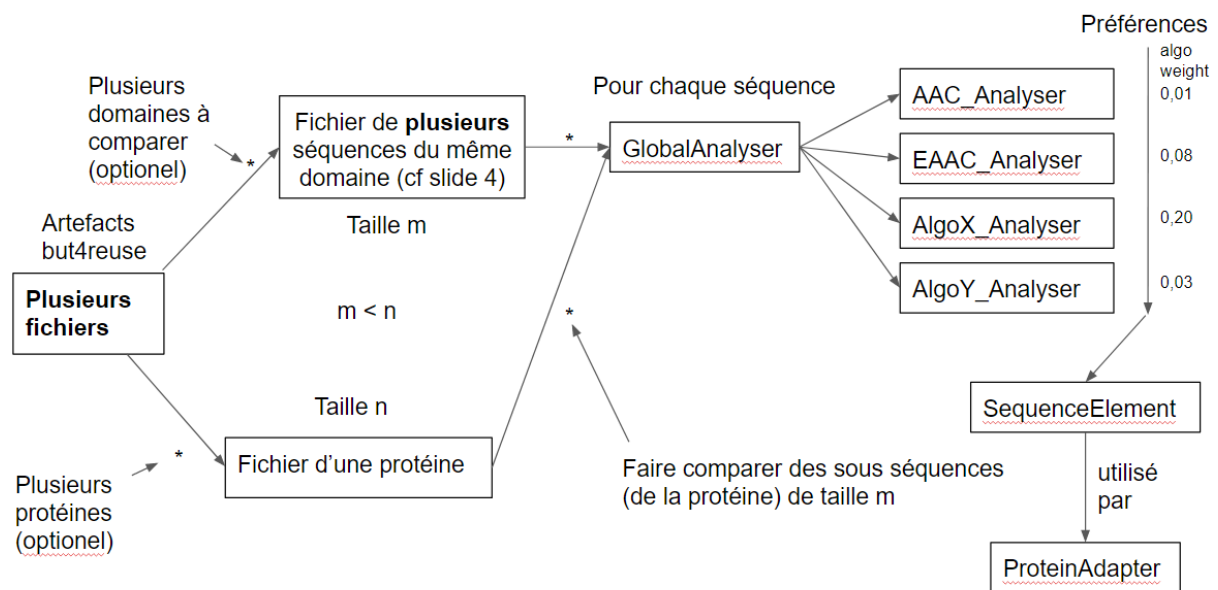
Pour partager notre travail, nous avons mis en place un Git privé (qui est un clone du Git but4reuse avec l'ajout de plugin protein), et utilisé l'IDE Eclipse Modeling 2018.

# Première idée développée pour vérifier l'appartenance d'une protéine à une famille de protéines

Nous avons, vers le début-milieu du projet, développé une idée bien différente de ce que nous avons présenté jusque là : plutôt que de chercher des similarités entre deux séquences, nous voulions à la place évaluer directement la similarité entre deux séquences.

En effet, nous sommes partis du principe qu'il fallait recouper d'une manière ou d'une autre les résultats de chaque type de descripteurs, de sorte à avoir une évaluation plus riche. De plus, nous voulions absolument pondérer les types de descripteurs, afin de leur donner plus ou moins de crédit dans l'évaluation de ces résultats.

Dans But4reuse, cela consiste en fait à définir un élément très englobant, à savoir la séquence elle-même (*SequenceElement*). Ainsi, la similarité de deux séquences est calculée, en réalisant une évaluation (pondérée) des types de descripteurs.



Nous commençons à avoir des résultats très satisfaisants pour savoir si une protéine appartenait à une famille de protéines, mais cela ne répondait pas vraiment au cahier des charges, à savoir de trouver des similarités entre protéines, et non "simplement" évaluer la similarité entre protéines.

La nature des résultats étaient simplement de voir que la similarité de deux séquences (valeur entre [0 ; 1]) était forte lorsque les protéines étaient de la même famille, et faible lorsqu'elles ne l'étaient pas.



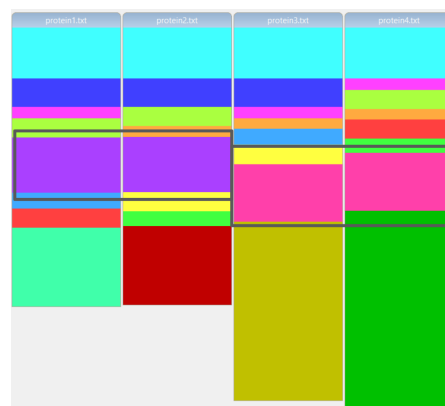
## Seconde idée développée pour vérifier l'appartenance d'une protéine à une famille de protéines

La seconde idée doit inévitablement se passer de la pondération des types de descripteurs.

De plus, il est préconisé de réaliser des tests sur un seul type de descripteurs à la fois : les tests concentrent plusieurs fichiers, et tester deux (ou plus) type de descripteurs va amener But4reuse à comparer des descripteurs de types différents, qui certes ne faussent pas les résultats (similarité égale à 0), mais complexifie fortement l'analyse (temps d'exécution nettement plus long).

Voici cependant un exemple où deux protéines d'une même famille, et deux protéines d'une autre famille, sont comparés entre elles, avec tous les descripteurs. On voit nettement deux gros blocs qui, pour chacun, contiennent les propriétés de la famille de protéines.

Remarque : si les deux autres protéines (3 et 4) n'avaient pas été ajoutées à l'identification, on comprend bien que le bloc de la famille (1 et 2) aurait été plus large.



Lorsqu'on compare des protéines d'une même famille, à savoir plusieurs fichiers de protéines, de même famille, c'est le bloc commun à chacune des protéines qui est plus intéressant : il décrit de quels descripteurs la famille est composée.

Cependant, parmi les descripteurs de ce bloc commun, en réalité sont retrouvés des caractéristiques assez communes, c'est à dire pas si représentative de la famille.

Ainsi, pour avoir de meilleurs résultats, il nous faut :

- un maximum de protéines d'une même famille, de sorte à préciser au mieux les caractéristiques (descripteurs) de la famille
- un maximum de protéines provenant d'autres familles, qui joue le rôle de "filtre" sur le bloc commun de la famille.

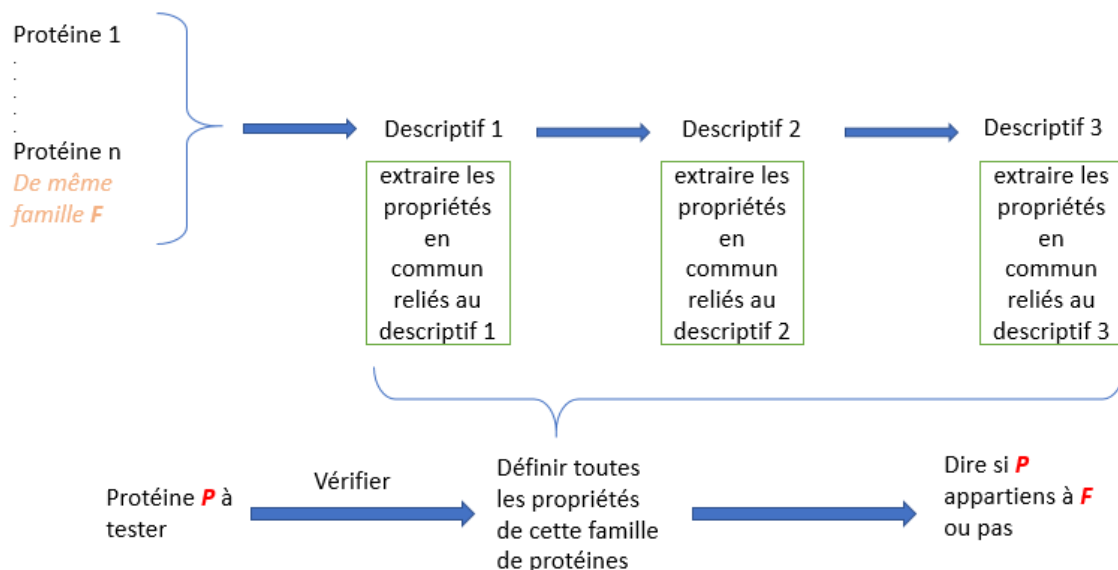
L'exemple du dessus montre partiellement ce phénomène, où le bloc cyan (bloc commun à toutes les protéines) se retrouverait effectivement dans les caractéristiques de la famille de la protéine 1 et 2, si les protéines 3 et 4 ne l'avaient pas "filtré".

Nous avons donc réalisé des tests un peu plus précis en essayant de s'inspirer de cette analyse, pour justement montrer qu'une protéine appartient à la même famille, plutôt que de simplement identifier les caractéristiques d'une famille :

L'idée de cette approche est de chercher les propriétés physiologique et biochimique en commun d'un ensemble de protéines, et de regarder si une protéine qu'on connaît pas si elle appartient à cette famille ou pas possède les même propriétés ou pas.

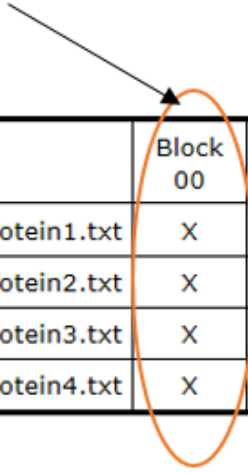
Plus que le nombre de protéines de cet ensemble est grand, plus les nombre de propriétés en commun est petit, ce qui nous permet de figurer les propriétés qui définissent cette famille et élimine les autres propriétés.

Le schéma suivant présente l'architecture globale de l'approche suivie:



En appliquant l'identification des blocs (feature identification) sur un ensemble de protéines qui contient un nombre de protéines intéressant, en sélectionnant dans les préférences un type de descripteur *D*, et dans le cas où on obtient un bloc en commun entre toutes les protéines de cet ensemble, on peut déduire que ce bloc en commun (qui est d'ailleurs, quand il existe, toujours le bloc 00 selon le principe de fonctionnement de but4reuse) correspond à une propriété qui définit la famille de protéine à laquelle toutes les protéines de cet ensemble appartiennent.

Toutes les  
protéines ont  
le bloc 00 en  
commun



	Block 00	Block 01	Block 02	Block 03	Block 04	Block 05	Block 06
protein1.txt	X	X	X	X		X	X
protein2.txt	X		X	X	X	X	
protein3.txt	X	X		X	X		
protein4.txt	X	X	X		X		X

Le fait qu'une nouvelle protéine ne partage pas cette propriété avec cet ensemble nous permet de déduire que cette nouvelle protéine n'appartient pas à cette famille.

	Block 00	Block 01	Block 02	Block 03	Block 04	Block 05	Block 06	Block 07	Block 08	Block 09	Block 10	Block 11	Block 12	Block 13	Block 14
PF00207_full_protein0.txt	X	X	X		X	X			X	X	X	X			
PF00207_full_protein1.txt	X	X	X	X	X		X	X	X		X	X	X	X	
PF00207_full_protein2.txt	X	X	X		X		X	X	X		X	X	X	X	
PF00207_full_protein3.txt	X	X	X	X	X				X	X				X	
PF00207_full_protein4.txt	X	X			X		X	X		X	X			X	
PF00207_full_protein5.txt	X	X	X	X		X		X	X				X		
PF00207_full_protein6.txt	X			X		X	X	X		X		X			
PF00207_full_protein7.txt	X			X		X	X	X		X		X			
PF00207_full_protein8.txt	X		X	X		X	X						X		
PF13173_full_protein0.txt		X	X	X	X	X					X		X	X	

La capture ci-dessus présente la matrice obtenue en appliquant les descripteur GTPC sur un ensemble de protéines d'une même famille PF0207 et une protéine d'une famille différente PF13173. Comme on peut remarquer, le block 00 est commun pour toutes les protéine de la famille PF0207 mais pas à la protéine de la famille PF13173 qui est à la dernière position.

Pour pouvoir décider, il faudra appliquer tous les descripteurs, un par un.

## Descripteur GTPC

	Block 00	Block 01	Block 02	Block 03	Block 04	Block 05	Block 06	Block 07	Block 08	Block 09	Block 10	Block 11	Block 12
PF00207_full_protein0.txt	X	X	X	X	X	X	X	X	X		X	X	X
PF00207_full_protein1.txt	X	X	X	X	X	X	X	X	X	X		X	X
PF00207_full_protein2.txt	X	X	X	X	X	X	X	X	X	X		X	X
PF00207_full_protein3.txt	X		X	X	X	X		X	X	X	X	X	X
PF00207_full_protein4.txt	X	X	X	X		X			X	X	X		X
PF00207_full_protein5.txt	X	X		X	X	X	X		X	X	X	X	
PF00207_full_protein6.txt	X	X	X	X	X	X	X	X		X	X		
PF00207_full_protein7.txt	X	X	X	X	X	X	X	X		X	X		
PF00207_full_protein8.txt	X	X	X	X	X	X	X	X	X	X	X	X	X
PF00207_full_protein9.txt	X	X	X		X	X	X	X	X	X			
PFxxxx.txt	X	X	X	X	X		X	X	X		X		

## Descripteur CKSAAP

	Block 00	Block 01	Block 02	Block 03	Block 04	Block 05	Block 06	Block 07	Block 08	Block 09	Block 10	Block 11	Block 12
PF00207_full_protein0.txt	X	X	X	X	X	X	X		X	X	X	X	X
PF00207_full_protein1.txt	X	X	X	X	X	X	X	X	X	X	X	X	X
PF00207_full_protein2.txt	X	X	X	X	X	X	X	X	X		X	X	X
PF00207_full_protein3.txt	X	X	X		X	X	X		X		X	X	X
PF00207_full_protein4.txt	X	X	X	X	X	X	X	X		X			X
PF00207_full_protein5.txt	X	X	X	X	X		X	X		X	X		
PF00207_full_protein6.txt	X	X	X	X		X	X	X		X			
PF00207_full_protein7.txt	X	X	X	X		X	X	X		X			
PF00207_full_protein8.txt	X	X		X	X			X	X			X	
PF00207_full_protein9.txt	X		X	X	X	X			X		X		
PFxxxx.txt	X	X	X		X	X		X	X	X	X	X	X

## Descripteur GDPC

	Block 00	Block 01	Block 02	Block 03	Block 04	Block 05	Block 06	Block 07	Block 08	Block 09	Block 10	Block 11
PF00207_full_protein0.txt	X			X					X	X		
PF00207_full_protein1.txt	X	X	X		X	X				X	X	X
PF00207_full_protein2.txt			X			X					X	
PF00207_full_protein3.txt		X				X	X					X
PF00207_full_protein4.txt								X				
PF00207_full_protein5.txt												
PF00207_full_protein6.txt	X	X	X	X	X		X					
PF00207_full_protein7.txt	X	X	X	X	X		X					
PF00207_full_protein8.txt									X			
PF00207_full_protein9.txt								X				
PFxxxx.txt								X				

## Descripteur TPC

	Block 00	Block 01	Block 02	Block 03	Block 04	Block 05	Block 06	Block 07	Block 08	Block 09	Block 10	Block 11	Block 12
PF00207_full_protein0.txt	X	X		X	X	X			X				
PF00207_full_protein1.txt	X	X	X	X	X	X	X		X	X	X	X	X
PF00207_full_protein2.txt	X	X		X	X	X	X		X	X	X		X
PF00207_full_protein3.txt	X	X	X	X	X		X	X	X	X	X	X	X
PF00207_full_protein4.txt			X					X		X			
PF00207_full_protein5.txt			X										
PF00207_full_protein6.txt						X						X	
PF00207_full_protein7.txt						X						X	
PF00207_full_protein8.txt	X	X	X		X		X	X			X		
PF00207_full_protein9.txt	X		X					X					X
PFxxxx.txt	X	X		X			X	X					

Jusqu'ici, on ne peut pas dire que la protéine PFxxxx n'appartient pas à la famille PF00207, car dans tous les cas où toutes les protéines de la famille PF00207 ont le bloc 00 en commun (descripteur GTCP- descripteur CKSAAP) la protéine PFxxxx avait, elle aussi, les mêmes éléments du bloc 00.

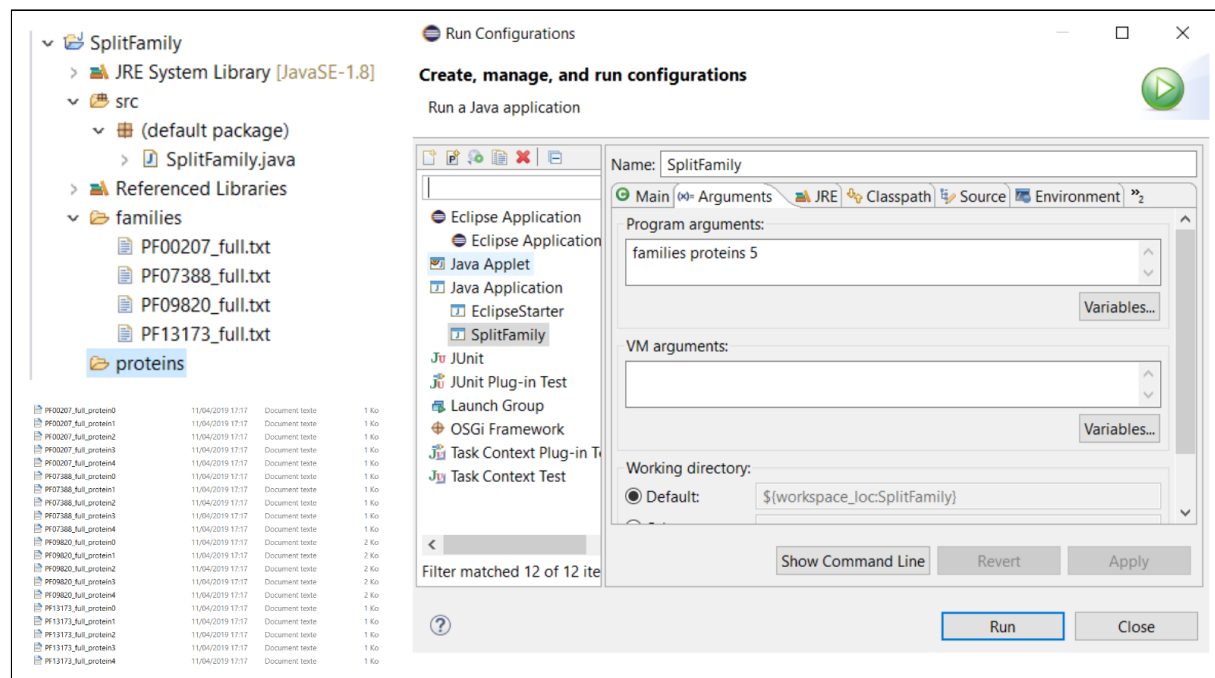
A la fin de l'application de tous les descripteurs, on peut déduire si la protéine PFxxxx appartient à la famille PF00207 ou pas.

# Diviser un fichier d'une famille de protéines, en plusieurs fichiers de protéines

Pendant notre projet, pour simplifier les tests que nous voulions réaliser, nous avons implémenté un petit programme *SplitFamily* externe à But4reuse qui produit plusieurs fichiers contenant chacun une seule protéine, à partir d'un fichier de la base de données Pfam contenant une famille de protéine.

Ce programme est alors paramétrable sur :

- le chemin du répertoire contenant les familles de protéines.
- le chemin du répertoire contenant les fichiers de protéines résultats.
- le nombre de protéines à extraire (un certain nombre > 0, ou *all*).



# Chronologie de notre travail

## Semaine 1

Installation de But4reuse, et application des tutoriels

## Semaine 2

Travail de compréhension sur les protéines et de la base de données Pfam

## Semaine 3

Implémentation d'un adaptateur selon la position et la nature d'un acide aminé dans une séquence : comme pour comparer les images, mais des lettres au lieu des couleurs.

Implémentation du parsing paramétrable des séquences ARN.

## Semaine 4

Implémentation des 6 premiers descripteurs présentés dans *iFeature* (AAC, EAAC, CKSAAP, DPC, TPC, DDE), mais sur l'idée de éléments de séquence (cf. Première idée développée pour vérifier l'appartenance d'une protéine à une famille de protéines.

## Semaine de vacances + semaine de partiel

Abandon de la première idée (car ne répond pas au cahier des charges du projet, et n'exploite pas vraiment les fonctionnalités de But4reuse), et implémentation des descripteurs sous forme de "petits éléments" comme présentés dans le rapport, pour ainsi rendre l'étude de similarité plus fine et détaillée.

## Semaine 5

Réorganisation du projet (plus de structure), ajout des préférences, et possibilité d'analyse multiple sur les séquences (différents types de descripteurs lancés en même temps).

## Semaine 6

Modifications sur la généricité des descripteurs (élément de base *DescriptorElement*, et plus de réutilisation de code sur certaines familles de descripteurs).

Ajout des descripteurs groupés (GAAC, EGAAC, CKSAAGP, GDPC, GTPC), à ce stade, il y a 11 type de descripteurs fonctionnels.

Ajout du programme externe *SplitFamily*.

## Semaine 8 + vacances + semaine 9

Moment à vide, seulement une grosse modification sur la notion de descripteurs groupés, pour plus de généricité (utilisation beaucoup plus large de descripteurs groupés), et l'ajout des descripteurs CTDC / CTDI (cas particulier de descripteur groupé / utilisation de la notion de groupes).

# Manuel d'utilisation But4reuse

Comme But4reuse est développé à base de plugin Eclipse, on l'exécute donc comme un programme Eclipse, et comme nous développons un adaptateur (et donc une extension), nous le lançons depuis Eclipse.

- *Right click* sur un des plugin But4reuse
- *Run As* → *Eclipse Application*

Une fois But4reuse ouvert, et si cela n'est pas déjà fait, il faut sélectionner la perspective But4reuse (et non Eclipse).

- *Open Perspective* (en haut à droite) → *BUT4Reuse*

Ensuite il s'agit de créer un projet But4reuse : il nous permet de définir des modèles d'artefacts, c'est-à-dire des ensembles d'artefacts à étudier.

- *File* → *New* → *Project*

Une fois un modèle créé, il faut bien évidemment lui ajouter des artefacts à comparer.

- *Drag & Drop* de fichiers depuis l'OS dans l'onglet *Input Drop*

De plus, et il s'agit d'un point essentiel de notre projet, il est possible de sélectionner les critères que l'on souhaite étudier pour la prochaine identification de blocs, dans les préférences. Ainsi, en lançant une identification de blocs (*feature identification*) sur un modèle, celui-ci se lancera en tenant compte des préférences définies préalablement.

- *Window* → *Preferences* → *BUT4Reuse* → *Adapters* → *Protein descriptors*

Pour modifier la marge d'acceptation de la similarité entre deux éléments, il faut modifier les préférences de similarité.

- *Window* → *Preferences* → *BUT4Reuse* → *Similarity*

Enfin, on peut lancer une identification de blocs sur un modèle

- *Right click* sur un *Artefact Model*
- *Feature identification*