UNIVERSITÉ DU QUÉBEC À MONTRÉAL

IDENTIFICATION ET CARACTÉRISATION DE MICROARNS DANS LES ESTs DU BLÉ PAR DES MÉTHODES BIOINFORMATIQUES

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

À LA MAÎTRISE EN INFORMATIQUE

PAR

MICKAEL LECLERCQ

AVRIL 2012

REMERCIEMENTS

Je présente mes sincères remerciements au professeur Abdoulaye Baniré Diallo, qui a supervisé mon avancement tout au long de ma maîtrise. Sa présence, sa sagesse, ses riches enseignements et ses conseils m'ont été d'une grande utilité tout au long de mes travaux. C'est aussi en partie grâce à son aide que j'ai l'honneur de pouvoir continuer mes études à McGill avec une bourse d'excellence, et lui en suis extrêmement reconnaissant.

Je remercie aussi le professeur Fathey Sarhan, pour son support dans mes demandes de bourses et d'admission pour mon doctorat. Par ailleurs, je remercie particulièrement Madame Zahra Agharbaoui, assistante de recherche du Pr Sarhan, qui m'a permis de conduire l'étude bioinformatique de toutes ses données. Durant cette maîtrise, j'ai partagé des moments d'intenses échanges et réflexions qui ont abouti aux résultats présentés ici.

Mes remerciements vont aussi au Professeur Vladimir Makarenkov et tous les membres du laboratoire de bioinformatique de l'UQAM, particulièrement Etienne Lord, pour sa patience et sa persévérance, qui m'a aidé à me lancer dans la programmation. Son soutien m'a souvent été indispensable pendant cette maîtrise. Alix Boc, pour m'avoir donné l'accès à ses outils de calcul, et notamment m'avoir appris à m'en servir. Dunarel Badescu, qui m'a donné le goût d'utiliser Linux, et d'en apprendre d'avantage sur ce système d'exploitation qui m'est devenu incontournable pour ce projet. Sachez que je n'oublierai jamais la bonne ambiance qui a toujours régné dans le laboratoire. Les discussions ont toujours été fort constructives, et l'expérience que chacun a bien voulu partager m'a beaucoup apporté.

Je n'oublie pas mes parents, sans lesquels je ne serais jamais arrivé à ce niveau sans les moyens qu'ils m'ont apportés.

Enfin tous ceux qui de près ou de loin ont contribué à la réalisation de ce mémoire, qu'ils en trouvent ici l'expression de mes sincères remerciements.

TABLE DES MATIERES

LISTE DES FIGURES

Figure Page

LISTE DES TABLEAUX

Tableau Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ADN	Acide DésoxyriboNucléique
ARN	Acide RiboNucléique
EST	Expressed Sequenced Tag
MFE	Energie minimale de repliement (*Minimum Folding Energie*)
NGS	Séquençage de nouvelle génération (*Next Generation Sequencing*)
pb	Paires de bases
pré-microARN	Précurseur du microARN
SVM	Machine à vecteurs de support (*Support Vector Machine*)

RÉSUMÉ

Les microARNs sont de petits ARNs qui participent à la régulation de l'expression génique dans les organismes vivants. Depuis l'apparition des nouvelles techniques de séquençage à haut débit, leur identification et caractérisation dans les espèces animales et végétales font partie des défis actuels de la bioinformatique. Dans ce mémoire nous abordons ce problème en recherchant les microARNs exprimés chez le blé sous une dizaine de conditions environnementales différentes. Contrairement aux études antérieures similaires dans d'autres organismes, des difficultés additionnelles se sont ajoutées, associées à la partialité du génome du blé disponible, la multitude de conditions expérimentales et le peu de microARNs connus retrouvés par prédiction. Une nouvelle approche, employant une double validation par deux algorithmes de prédiction, a permis d'identifier plus de 3862 microARNs potentiels chez le blé ainsi que leurs gènes cibles. Parmi eux, 206 sont différentiellement exprimés entre les conditions expérimentales. Ces microARNs ont été répartis en 1222 familles en fonction des paramètres de leur similarité intra et intergroupe déduite des microARNs connus de mirBase. Pour minimiser le nombre de faux positifs, nous avons développé une méthode d'apprentissage machine (*miRdup*) pour valider la position des microARNs séquencés sur son pré-microARN en estimant plusieurs caractéristiques associées à ces petits ARNs. Ce dernier nous a permis d'établir 1016 microARNs avec un haut score de prédiction (502 familles). Pour chacun des microARNs prédits, en exploitant les données de leurs niveaux d'expression et l'identification des gènes ciblés, nous avons décrypté leurs rôles dans les différentes conditions expérimentales imposées au blé. Plusieurs microARNs sont en cours de validation expérimentale. Par ailleurs, pour faciliter l'implémentation de cette plateforme, nous avons intégré le pipeline de recherche conçu au cours de ce mémoire dans le logiciel *Armadillo*. Ce dernier permettra à l'avenir de faciliter la reproduction d'une telle étude chez d'autres plantes.

Mots clés : microARNS, prédiction, séquençage haut débit, blé, apprentissage machine.

INTRODUCTION

La recherche porte un grand intérêt aux microARNs depuis la découverte de leur capacité de régulation par ciblage de l'ARN (Ambros 1989, Ruvkun 2001). En effet, les microARNs participent activement à la régulation post-transcriptionnelle (Swami 2010) et leur activité serait à l'origine des résistances diverses aux stress environnementaux observées chez les plantes (Sunkar et al. 2006). De façon générale, un microARN est contenu dans une séquence plus longue, appelée le précurseur, ou pré-microARN, qui se replie en structure secondaire sous la forme d'une épingle à cheveux. Ce microARN, une fois détaché de son précurseur, a la capacité de bloquer la traduction d'un ARN messager cible, empêchant ainsi la production de la protéine de ce dernier. Due à leur importance grandissante, des bases de données de microARNs ont été créées pour les répertorier. La principale est miRbase (Griffiths-Jones et al. 2006), qui contient actuellement les microARNs publiés et dont la dernière version comporte près de 17000 microARNs répartis dans environ 150 espèces. Elle peut être associée à miRMaid (Jacobsen et al. 2010), clonée à partir de miRbase et qui possède en plus une interface plus conviviale afin de créer des requêtes complexes d'analyse. PMRD est une autre base de données, spécialisée dans les microARNs des plantes (Zhang et al. 2010). Elle répertorie plus de microARNs découverts par prédiction informatique que miRbase. Enfin, microRNA.org (Betel et al. 2008) se concentre sur les microARNs de cinq espèces seulement et expose leurs cibles et profils d'expression sous des conditions variées.

L'étude décrite dans ce document, dont les résultats sont exposés principalement dans le chapitre III, repose sur des données de séquençage issues de *Triticum aestivum*, le blé commun, soumis à différentes conditions de stress. Ce blé est une cible de choix, car il fait partie de la nourriture de base consommée dans le monde. L'augmentation de la population mondiale est un défi pour l'humanité où la nécessité de subvenir aux besoins alimentaires

passe par l'amélioration des plantes les plus consommées afin d'en tirer le meilleur rendement. D'un point de vue économique, le Canada est un des exportateurs les plus importants de la planète, et de nombreuses étendues de terres restent inexploitées dues aux conditions peu optimales pour l'agriculture. Enfin, le froid et la présence de sel ou d'aluminium dans certains sols ne permettent pas à la plante de se développer correctement.

Le génome séquencé du blé n'est pas encore disponible. Étant hexaploïde, son séquençage reste un grand défi pour la communauté scientifique. Ainsi, en attendant, d'autres voies doivent être exploitées pour découvrir la façon dont ses gènes sont régulés. L'identification, la caractérisation et la régulation des gènes du blé restent très peu connus. Actuellement, seulement une quarantaine a été identifiée et mise dans les banques de données. Nous apportons une nouvelle connaissance sur cette régulation à travers l'identification des microARNs associés à différents stress abiotiques.

Dans ce mémoire, nous proposons une approche bioinformatique pour prédire les microARNs obtenus à partir d'un séquençage AB SOLID de plusieurs millions de fragments de courts ARNs. L'approche proposée est basée sur un génome partiel du blé, constitué seulement d'ESTs, et emploie une validation croisée entre les résultats issus de différents algorithmes de prédiction de potentiels précurseurs contenant les principales caractéristiques connues pour ce type de structure. Le repliement permet d'obtenir une structure secondaire testée ensuite par des algorithmes d'apprentissage. Les précurseurs prédits en commun par deux programmes de prédiction sont retenus et leurs microARNs soumis à une recherche de gènes cibles. Une fois les microARNs caractérisés, une analyse de leurs expressions différentielles dans les principales conditions de stress est réalisée par une procédure statistique dérivée des analyses de micropuces. Enfin, les annotations fonctionnelles des microARNs prédits ont été réalisées en exploitant la puissance de l'ontologie de Gènes GO.

Ce mémoire présente donc la première étude exhaustive des petits ARNs issus du blé sous différentes conditions de stress. Il est divisé en trois chapitres, une conclusion et trois annexes :

Le chapitre I décrit les principes de base de la génomique suivi de la biologie du microARNs.

Le chapitre II expose brièvement les méthodes de séquençage et les logiciels utilisés pour les différentes étapes de la prédiction de microARNs. Il présente entre autres les méthodes d'alignement des fragments séquencés de courts ARNs contre un génome de référence, ceux de repliements de pré-microARNs en structures secondaires, les approches de prédictions des structures de précurseurs capables d'induire un microARN et ceux pour l'identification de gènes pouvant être ciblés par un microARN.

Le chapitre III est divisé en deux parties. La première partie présente les deux principales approches connues pour découvrir de nouveaux microARNs à partir d'un séquençage à haut débit. Puis, la seconde partie décrit notre nouvelle approche de prédiction et de caractérisation de microARNs à partir de séquençage à haut débit et différents stress abiotiques. Les principaux résultats obtenus et une discussion de ceux-ci sont également présentés.

Ce chapitre est suivi d'une conclusion de l'étude menée durant cette maîtrise. Celle-ci établit une synthèse du travail qui a été effectué, basée sur les résultats obtenus par l'étude et les perspectives de recherche qui en découlent.

Ce mémoire comporte trois annexes. L'Annexe A présente la plateforme d'automatisation de tâches *Armadillo*, dans laquelle j'ai participé au développement. Elle intègre nos étapes de recherche sous la forme d'un flux de travail graphique (*Workflow*), facilitant ainsi la recherche de nouveaux microARNs. L'Annexe B présente *mirDup*, notre méthode basée sur les techniques d'apprentissage machine pour valider le duplex microRNA et précurseur. L'Annexe C contient tous les codes sources écrits dans le cadre de l'étude.

CHAPITRE I

PRINCIPES DE GÉNOMIQUE ET MICROARNS

1.1 Principes de base de la génomique et de l'évolution

L'information génétique portée par l'ADN est le constituant même de la vie, d'autant plus qu'il est le support essentiel de l'hérédité, car transmis de génération en génération (par accouplement, mitose ou scission). Au cours du temps, bien que les origines de la vie reposant sur l'apparition des premières molécules organiques sur Terre restent incertaines, il n'en demeure pas moins que la vie a évolué sous un panel de conditions environnementales très différentes. En effet, plusieurs chercheurs avancent que les premières formes de génome étaient sous forme d'ARN (Acide RiboNucléique) (Gilbert 1986). La plasticité des molécules constituant le génome et ses procédés de réplication a alors entrainé une adaptation des organismes grâce à des mutations et des échanges entre ceux-ci. Cela a abouti à la diversité des êtres vivants actuellement présents dans nos écosystèmes. Depuis Darwin et ses premiers travaux sur l'origine des espèces publiés en 1859 (Darwin 1859), l'homme a cherché à classifier les espèces qu'il a découvertes. Différentes classifications ont alors été élaborées, dont la dernière en date qui regroupe les archées, bactéries et eucaryotes. Les eucaryotes sont plus communément appelés en fonction de leur regroupement comme les animaux, les plantes et les protistes (Cavalier-Smith 2004).

L'ADN et l'ARN sont constitués de quatre bases azotées : Adénine, Guanine, Thymine (remplacée par l'Uracile dans l'ARN), et Cytosine. Ces bases alignées forment de longues

séquences dont des morceaux précis permettent l'attachement d'ARN polymérases sur l'ADN. Cette dernière traduit l'ADN en de nouvelles molécules d'ARN, nommés ARN messager. L'ARN messager est lui aussi décortiqué par des ribosomes qui vont lire les nucléotides trois par trois, donnant des combinaisons de nucléotides appelées de codons. Chaque codon détermine un acide aminé, et leur assemblage bout à bout forme des chaines d'acides aminés. Par de complexes séries de repliements et modifications, ces chaines sont utilisées pour produire toutes les protéines d'un organisme vivant.

Ainsi, chaque organisme vivant est défini par son génome, qui représente l'ensemble du matériel génétique d'un individu ou d'une espèce particulière, codé sous forme d'ADN (majorité des espèces) ou d'ARN. La taille d'un génome peut varier de quelques milliers de bases pour des organismes unicellulaires simples ou des virus, jusqu'à des milliards pour des êtres beaucoup plus complexes. Cette taille ne reflète pas la complexité de l'organisme apparente, où certains êtres unicellulaires comme l'amibe, *Amoeba nubia,* ou des plantes telles que *Paris japonica* ont des tailles de génome 50 à 200 fois plus grandes que celui de l'humain (Pellicer et al. 2010).

La structure d'un chromosome se présente comme montré à la Figure 1.1. Les bases azotées alignées les unes après les autres composent l'ADN, qui lui-même porte des gènes. Chaque gène contient des exons, les parties codantes, et des introns, parties non codantes. L'ADN est replié autour de protéines, les histones, regroupées plusieurs fois en petit nombre afin de former des nucléosomes. Au final, le repliement des nucléosomes s'effectue en colonnes, aboutissant au chromosome.

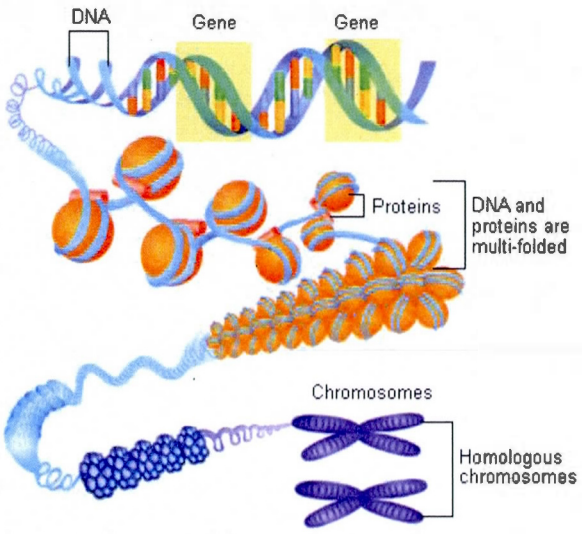


Figure 1.1 Structure du chromosome

(Alberts 2002)

La Figure 1.2 résume les étapes de la fabrication d'une protéine à partir de l'ADN dans une cellule eucaryote. Dans le noyau, l'ADN est transcrit en pré-ARN messager par une ARN polymérase, non schématisée sur la figure. Le processus d'épissage s'effectue ensuite sur le pré-ARN aboutissant à un ARN messager mature, qui est transporté dans le cytoplasme de la cellule. Ce dernier sera traduit par des ribosomes afin d'obtenir des chaines d'acides aminés, ou polypeptides, qui vont préalablement subir une cascade de transformations chimiques, appelée maturation des protéines, afin de devenir des protéines fonctionnelles.

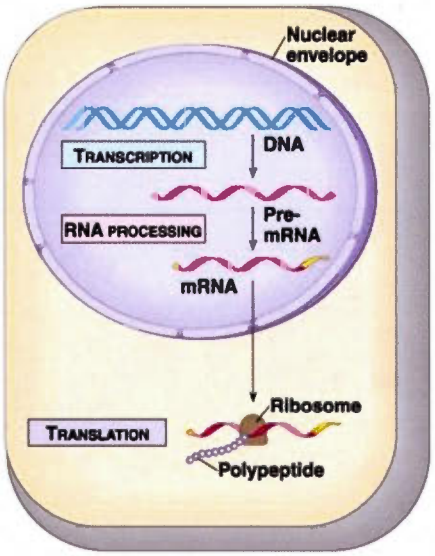


Figure 1.2 Étapes de transformation de l'ADN à la protéine

(Alberts 2002)

1.2 Génome et ESTs du blé

Pour pouvoir analyser un génome il faut le séquencer au préalable. C'est-à-dire rechercher l'ensemble des nucléotides qui le composent et leur emplacement sur les chromosomes (voir section 0 pour plus de détails). Cependant, dues aux limites des techniques de séquençage, seules des séquences, ou morceaux de séquences, dont la taille est inférieure à 2000pb peuvent être produits à la fois. Un assemblage est alors nécessaire. Cependant l'assemblage est un processus très complexe. Car les génomes contiennent de larges zones répétées ou contiennent trop de copies des chromosomes (on parle alors de polyploïdie). Il faut noter que plusieurs espèces de blé ont entre quatre à six paires de chromosomes. Ainsi, pour accélérer la recherche au niveau de la génomique fonctionnelle du blé, les chercheurs ont fait séquencer plusieurs milliers d'ESTs (*Expressed Sequence Tags*).

Les ESTs sont obtenus à partir d'ADN complémentaire. Ils sont moins chers à obtenir puisqu'ils proviennent directement d'ARNs exprimés soumis à une transcriptase inverse. En

effet, cette dernière permet le retour de l'état ARN à ADN. Ils sont essentiellement constitués d'exons, les séquences codantes, puisque l'épissage de l'ARN a retiré les introns. Ainsi, les ESTs ne sont qu'une vision partielle des zones du génome d'où ils proviennent.

Dans le cas du génome complet du blé, il a été complètement séquencé, mais n'est pas encore assemblé et à peine annoté. Pour notre étude, nous avons regroupé les ESTs de plusieurs banques de données spécialisées du blé, telles que *GrainGenes* (wheat.pw.usda.gov/ggpages/genomics.shtml), *TIGR* (blast.jcvi.org/euk-blast/index.cgi?project=tae1) (Pertea et al. 2003), *WheatDB* (wheatdb.ucdavis.edu:8080/wheatdb/), *TAGI* (compbio.dfci.harvard.edu/tgi/), *Komugi* (www.shigen.nig.ac.jp/wheat/komugi/) et une base de données locale provenant de notre laboratoire de recherche. Après les avoir regroupées et concaténées, les doublons ont été éliminés. Chaque EST contient un numéro d'accession. En plus, étant donné les différentes banques de données, plusieurs séquences peuvent être très similaires ou incluses dans des séquences plus grandes. Actuellement, plus d'1.4 millions d'ESTs contenant près de 860 millions de bases ont été rassemblées.

1.3 Biologie des acides ribonucléiques (ARNs)

Tout comme l'ADN double brin, les ARNs simples brins sont composés de longues chaines nucléotidiques. Chaque nucléotide est constitué d'une base azotée, d'un sucre ribose, et d'un groupe phosphate. L'ARN est produit par la transcription de l'ADN dans le noyau. Il est composé des mêmes bases azotées que l'ADN excepté l'uracile qui remplace la thymine. L'uracile possède les mêmes propriétés de liaisons envers l'adénine que la thymine. Selon sa taille, l'ARN est capable de se replier sur lui-même grâce à des liaisons chimiques qui mènent à des structures secondaires et tertiaires particulières. Ces dernières confèrent un pouvoir fonctionnel à une partie d'entre eux.

Il existe plusieurs types d'ARNs (voir Tableau 1.), les ARN messagers, codants pour les protéines, constituent le principal type et sont communs à tous les organismes vivants et certains virus. D'autres types d'ARN sont impliqués dans la régulation post-transcriptionelle

de l'expression de gènes, la réplication de l'ADN et de nombreuses autres fonctions régulatrices.

Tableau 1.1 Liste des principaux types d'ARN

Type	Abréviation	Fonction	Distribution	Références
ARNm	ARN messager	Code pour la protéine	Tous les organismes	
ARNr	ARN ribosomal	Traduction	Tous les organismes	
ARNt	ARN de transfert	Traduction	Tous les organismes	
ARNtm	ARN (Transfer-messenger RNA)	Recyclage des ribosomes bloqués	Procaryotes	(Zwieb & Wower 2000)
ARNsn	Petit ARN nucléaire (Small nuclear RNA)	Épissage, maintien des télomères, régulation de facteurs de transcription	Eucaryotes	(Birnstiel & Schaufele 1988)
ARNsno	Petit ARN nucléolaire (Small nucleolar RNAs)	Modification des ARNs (ex : méthylation)	Eucaryotes	(Kiss 2001)
ARNpi	ARN piwi (Piwi-interacting RNA)	Epigénétique, bloquent les rétrotransposons	Animaux	(Seto et al. 2007)
ARNsi	ARN interférents	Régulation post-transcriptionnelle	Eucaryotes	(Hamilton & Baulcombe 1999)
miARN	microARNs	Régulation post-transcriptionnelle	Eucaryotes	(Lee et al. 1993)

Plusieurs ARNs ont des fonctions qui découlent directement de leurs propriétés structurales. Par exemple les ARNs de transfert ont une structure en trèfle comme présentée à la Figure 1.3. En se repliant, la molécule d'ARN forme une structure stable maintenue par des ponts hydrogènes entre les paires de bases A-U, C-G, G-U. Les paires G-C et A-U sont nommées Watson-Crick (Watson & Crick 1953), et la paire G-U est appelée *Wobble*. Au niveau de la stabilité, les liaisons G-C sont plus fortes que les liaisons A-U qui sont plus robustes que G-U. Plusieurs éléments structuraux peuvent coexister dans une structure secondaire

(Figure 1.3) : Les boucles internes, les boucles terminales (loop), les empilements, les hernies (bulges) et les boucles multiples (Dardel & KÃ©pÃ).

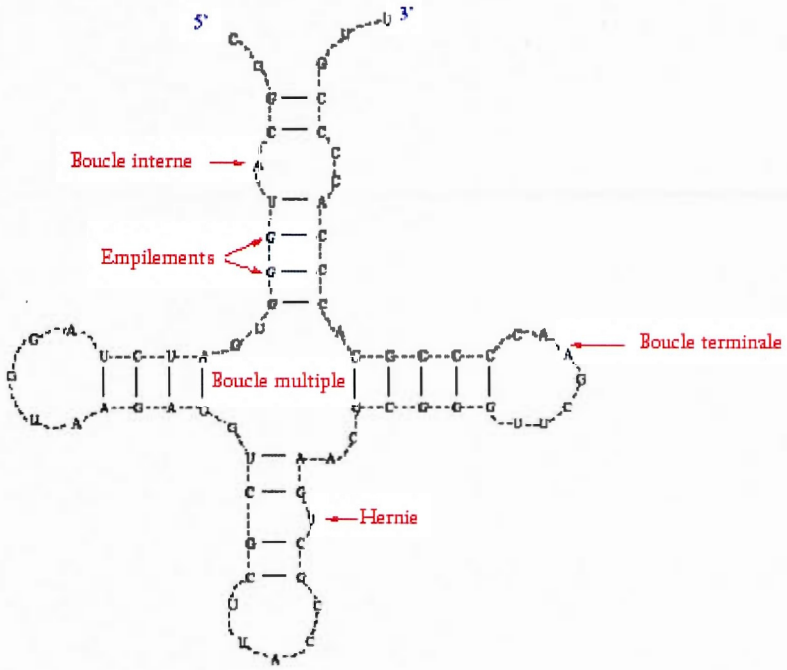
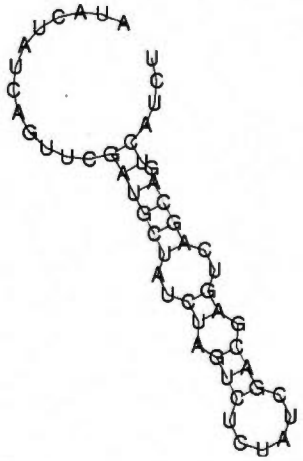


Figure 1.3 Éléments structuraux de la structure secondaire de l'ARN

Une structure d'ARN peut être représentée de façon visuelle ou encodée en format *parenthésé* (voir Table 1.2) en utilisant le format suivant : (1) une paire de parenthèse correspond à une liaison entre deux nucléotides, (2) et un point correspond à un nucléotide qui n'a pas d'appariement.

Table 1.2 Exemple de représentation d'une structure secondaire d'un ARN

Séquence	AUACUAUCAGUUCGAUGCUAUCUAGUCUCUAUCGACGAGUCAGCAGUCAUCU
Représentation parenthésée	..........((((((..((.(((.....))).)).)..)))).))....
Représentation graphique	

1.4 Biologie des micro acides ribonucléiques (microARNs)

Les microARNs sont des molécules simples brin dont la longueur varie entre 15 et 35 nucléotides, la majorité de ceux connus ont 21 nucléotides. Ils ont été découverts en 1993 chez *C. elegans*, un nématode, (Lee et al. 1993) et ont été formellement appelés microARNs en 2001 (Lagos-Quintana et al. 2001). Ils ont la particularité d'être conservés entre les espèces au cours de l'évolution, dénotant une pression sélective positive. Ils sont localisés autant dans les introns que les exons, voir même dans les régions intergéniques (Lu et al. 2005). Ils régulent environ 10 à 30% des gènes codant d'un organisme (Paranjape et al. 2009) en inhibant la traduction d'un ARN messager ciblé. Un microARN a la capacité d'inhiber la transcription de plusieurs gènes et plusieurs microARNs peuvent cibler le même gène. Dépendamment du type d'espèce (plante ou animal), les microARNs ont des mécanismes d'attache à leurs cibles différents. Il est connu que l'identification de l'appariement est plus simple chez les plantes que chez les animaux, car la complémentarité est quasi parfaite chez

les végétaux et les algorithmes de prédiction de gènes cibles tiennent compte de cette différence.

1.4.1 Biogenèse des microARNs

La Figure 1.4 présente un schéma de la synthèse des microARNs. Dans le noyau, un gène qui code pour un microARN est transcrit par une ADN polymérase II ou III, en pri-miRNA. Il est ensuite scindé par le complexe protéique qui associe une RNase (ribonucléase) de type Drosha III et DGCR8 (protéine de fixation des ARNs, appelée aussi Pasha), menant ainsi à un précurseur. Ce dernier est exporté du noyau vers le cytoplasme où il est débarrassé de sa boucle ARN par Dicer (hélicase avec motif ARN) et le cofacteur TRBP pour générer un duplexe microARN:microARN* (microARN étoile) d'environ 21 nucléotides libéré dans le cytoplasme. Le microARN* correspond à la séquence complémentaire du microARN sur l'épingle à cheveux, avec un décalage d'environ 2 nucléotides en plus en 3'. Un des deux brins, le microARN, est alors incorporé dans le complexe RISC (miRNA induced silencing complex) qui va le guider vers un ARN messager cible et réprimer son expression. Cette dernière étape peut s'effectuer par un clivage de l'ARNm ciblé s'il y a une complémentarité parfaite entre le microARN et l'ARNm, ou par répression traductionnelle si la complémentarité est imparfaite (Lamoril et al. 2010).

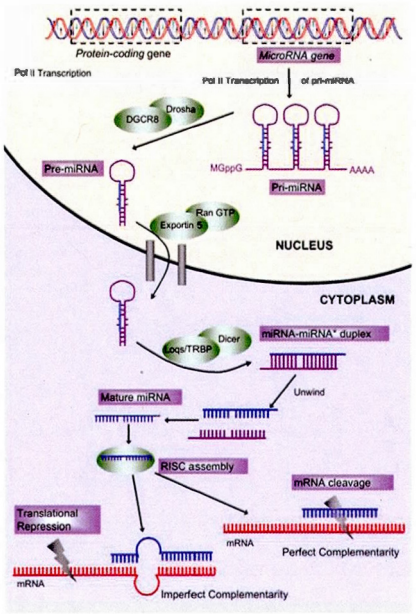


Figure 1.4 Biogenèse des microARNs

(Kadri et al. 2009)

1.4.2 Structure des microARNs

Les microARNs se distinguent par une structure secondaire particulière lors du repliement de l'ARN. La Figure 1.5 montre un microARN (en rouge) qui repose sur un précurseur. Ce précurseur forme une épingle à cheveu, avec deux branches, une boucle au bout, et de multiples hernies le long des branches. Ce cas s'explique par la présence d'un motif répété dont la répétition inversée mène à un appariement antiparallèle. Cette structure se calcule par des algorithmes de repliement spécialisés (voir section 2.4) qui reposent sur les interactions chimiques des bases appariées.

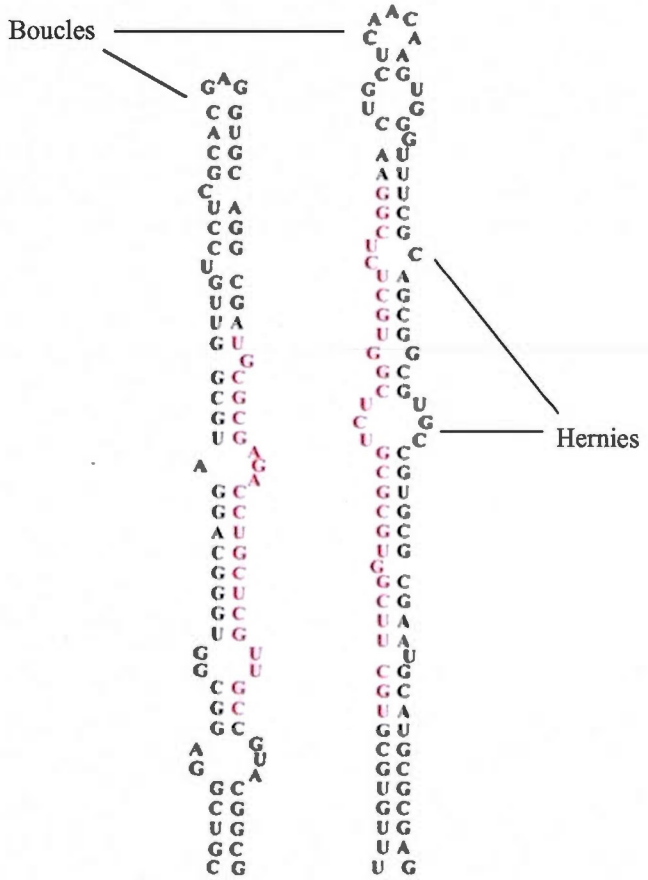


Figure 1.5 Structures secondaires des précurseurs

(Grey et al. 2005)

1.4.3 Classification des microARNs

Chaque espèce contient un nombre varié de microARNs. Plusieurs d'entre eux sont conservés entre les espèces par l'évolution. Afin de retracer cette conservation, les chercheurs suivent une nomenclature précise (Griffiths-Jones et al. 2008). Un microARN est dit conservé lorsqu'il est présent chez plusieurs espèces avec une séquence homologue. La conservation existe aussi pour un microARN qui possède le même précurseur et une séquence légèrement différente. La classification des microARNs sert donc essentiellement à

discerner ceux qui sont conservés entre les espèces et regrouper les séquences similaires ayant potentiellement des cibles équivalentes. Le choix de nommer et numéroter un nouveau microARN est effectué par *mirBase* (Griffiths-Jones 2004). À partir d'un alignement de la séquence sur la base de données Rfam (Griffiths-Jones et al. 2005), le consortium attribue une famille. Leur nomenclature s'effectue de la façon suivante : les séquences commencent toujours par un nom abrégé d'une espèce (ex : *hsa* pour *homo sapiens*), suivit par "*mir*", puis d'un chiffre qui représente la famille et s'incrémente par ordre de découverte. Enfin, une dernière lettre {a-z} représentant un membre de la même famille est rajoutée (exemple : hsa-mir-33a).

1.4.4 Fonctions des microARNs

L'un des grands enjeux de la recherche de nouveaux microARNs est de pouvoir exploiter leur faculté de cibler des ARNm pour altérer le mécanisme de fonctionnement des espèces (exemple : améliorer la résistance du blé au froid). Leur capacité de blocage post-transcriptionnel influence de nombreuses fonctions biologiques chez les animaux et les plantes, présentées dans cette section. A moyen ou long terme, ils pourraient devenir de formidables outils en biologie moléculaire ou en médecine, que ce soit pour la recherche ou pour des traitements médicaux.

1.4.4.1 *Rôle des microARNs en physiologie humaine*

Le rôle physiologique des microARNs a été mis en évidence dans de nombreux organes ou tissus. Dans les cellules souches, une quarantaine de microARNs est spécifiquement exprimé et joue un rôle dans la maintenance de la pluripotence, le degré de différenciation cellulaire et plusieurs autres domaines (Suh et al. 2004). Dans les ovocytes, ils participeraient à la maturation de la lignée germinale (Murchison et al. 2007). Au niveau du cycle cellulaire, plusieurs études sur le cancer mentionnent une régulation par les microARNs à différents stades de ce cycle (Chalfie et al. 1981, Murchison et al. 2007), (Takamizawa et al. 2004), où certains cancers étaient associés à des expressions modifiées (plus faibles ou plus fortes que la normale) de microARNs. Ils agissent dans les muscles squelettiques et cardiaques pour

permettre leur croissance ou leur différentiation (Chen et al. 2006, Williams et al. 2009). L'hématopoïèse et l'immunité sont aussi soumises à la régulation des microARNs (Lindsay 2008, Xiao & Rajewsky 2009), ainsi que les tissus nerveux (Cao et al. 2006). Par ailleurs, plusieurs sources récentes indiquent que des microARNs de plantes ingérés par un animal peuvent y réguler l'expression de certains gènes, et donc avoir un impact direct du point de vue nutritionnel (Vaucheret & Chupeau 2012, Zhang et al. 2012).

Un exemple concret est décrit dans l'étude de (Cohen & Brennecke 2006), qui montre l'implication directe d'un microARN particulier, le miR-430, dans le développement embryonnaire chez le zebrafish. Ils ont observé que sa présence permet de réguler la transition de la transcription de l'ARNm maternel vers le zygote par dégradation. En l'absence du miR-430, l'ARNm maternel s'accumule anormalement, entrainant une interférence avec la morphogenèse.

1.4.4.2 *Rôle des microARNs en pathogenèse*

D'un point de vue pathologique, le rôle des microARNs est immense, tant les études sur le sujet sont diverses et où l'on découvre chaque jour de nouvelles pathologies dans lesquelles des microARNs sont impliqués (Castanotto & Rossi 2009, Cooper et al. 2009). Une pathologie impliquant des microARNs peut soit refléter la perte de leur fonction (Miska et al. 2007) ou leur surexpression (Clop et al. 2006). Cela s'explique par des mutations qui apparaissent dans un gène régulé par des microARNs ou au niveau du microARN en lui-même. Dans le premier cas, la mutation empêche l'attachement du microARN sur le gène cible et dans le deuxième cas, le microARN se retrouve à cibler un autre gène ou devient simplement non fonctionnel.

Une importante part des articles sur le sujet se penche principalement sur le rôle que peuvent jouer les microARNs dans les cancers, où des chercheurs ont pointé précisément leurs différentes implications pro et anti-prolifération, pro et anti-apoptotique, pro et anti-angiogenèse, ou encore pro et anti-métastasiques, dont chacune regroupe un ou plusieurs microARNs selon le type de cancer et sa localisation (Ruan et al. 2009). Récemment, c'est dans les maladies cardiovasculaires que des microARNs ont été détectés à des concentrations

anormales (Castanotto & Rossi 2009, Van Rooij et al. 2006), ce qui, à terme, pourrait servir de marqueurs biologiques renseignant sur de futures complications cardiaques. Aussi, ils ont un impact sur le développement des cellules du système nerveux central (Babak et al. 2004, Castanotto & Rossi 2009). Au niveau des maladies infectieuses, les virus ont la capacité d'utiliser la machinerie cellulaire et produisent aussi des microARNs qui influent directement sur la cellule (Nelson 2007, Pfeffer et al. 2005, Sarnow et al. 2006).

1.4.4.3 *Rôle des microARNs chez les plantes*

Les microARNs sont des régulateurs clés dans de nombreux processus biologiques des plantes (Zhang et al. 2007, Zhang et al. 2006), comme le développement des racines (Boualem et al. 2008, Guo et al. 2005, Wang, Wang, Mao,), des vaisseaux (Kim et al. 2005, Zhou et al. 2007), des fleurs (Chen 2004, Nag et al. 2009, Wang et al. 2009), des feuilles (Palatnik et al. 2003) et de leur morphogenèse (Lauter et al. 2005). Ils interviennent aussi durant le passage du stade végétatif au reproductif et ont un rôle crucial dans les réponses aux stress (Fujii et al. 2005, Hsieh et al. 2009, Navarro et al. 2006, Sunkar et al. 2006).

Un exemple dans la Figure 1.6 représente un aperçu de quelques familles de microARNs, indiqués par les numéros 156 à 408, identifiables dans miRbase. Ces microARNs ont une influence sur des gènes tels que AGO ou TCP, impliqués dans de nombreux processus biologiques comme la réponse au stress et à l'auxine, la floraison, le développement des feuilles, la régulation des microARNs, et la polarité des feuilles. Une même famille peut cibler plusieurs gènes, qui sont eux-mêmes impliqués dans plusieurs fonctions, révélant alors la complexité du réseau d'interactions possibles lors de la régulation des gènes par les microARNs.

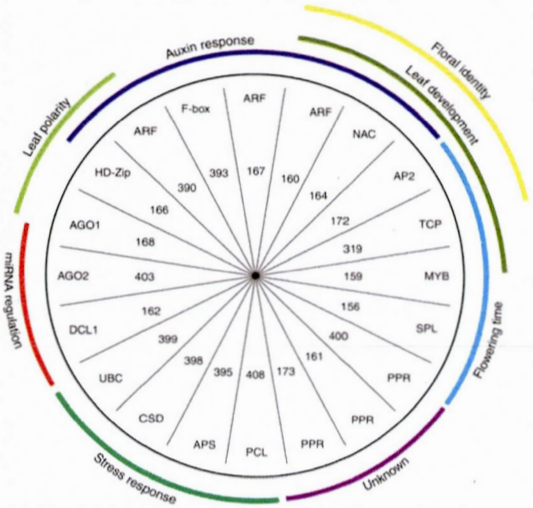


Figure 1.6 Fonction de quelques familles de microARNs chez les plantes

(Mallory & Vaucheret 2006)

CHAPITRE II

MÉTHODES DE SÉQUENÇAGE, ALIGNEMENT ET PRÉDICTION DES MICROARNS

L'identification de microARNs s'effectue en plusieurs étapes, dans un ordre précis. Des petits ARNs issus d'un séquençage sont alignés contre un génome de référence afin de localiser et d'extraire des précurseurs potentiels de microARNs. Ensuite, la structure secondaire des précurseurs est prédite, puis comparée avec des précurseurs validés expérimentalement afin de déterminer si elle a les mêmes caractéristiques qu'une vraie structure de précurseur. Enfin, les gènes ciblés par des microARNs portés par des précurseurs prédits sont recherchés par une comparaison de motifs d'affinité. Chaque étape peut s'effectuer de plusieurs manières. Ce chapitre explore de nombreux logiciels et approches existantes pour les différentes étapes, avec leurs avantages et inconvénients.

Dans ce chapitre, nous comparons brièvement les techniques permettant de séquencer l'ADN, de la fin des années 70 à aujourd'hui. Plusieurs caractéristiques sont mises en avant, telles que l'approche technique employée, la longueur des séquences lues par les séquenceurs, et les coûts et la vitesse d'opération d'un séquençage. Nous décrivons aussi l'organisation des résultats bruts issus de la plateforme AB SOLID qui a été utilisée pour générer les données de l'étude sur laquelle porte ce mémoire. Par la suite, nous abordons les défis que soulève le problème de l'alignement des séquences contre un génome de référence ou des ESTs. Une

dizaine de logiciels d'alignement existent dans la littérature, nous avons choisi de décrire le fonctionnement et les résultats bruts de trois d'entre eux : *MAQ* (Li, Ruan & Durbin), *Bowtie* (Langmead et al. 2009) et *SHRiMP* (Rumble et al. 2009). Ensuite, afin de prédire les structures des pré-miRNAs, une description de l'algorithme de repliement de séquences d'ARN est donnée. Puis, les méthodes employées et la structure des résultats des deux logiciels de prédiction de microARNs qui sont utilisés dans cette étude sont décrites. Enfin, nous expliquons l'approche de recherche des gènes ciblés par les microARNs avec le logiciel *TAPIR*.

Pour faciliter la compréhension du lecteur, tout le long de ce chapitre nous suivons l'évolution d'un fragment issu du séquençage des petits ARNs du blé, le fragment 1557_1102_758. Il permettra aussi d'avoir un aperçu de l'évolution des données au cours du processus de prédiction des microARNs.

2.1 Séquençage

Le séquençage consiste à déterminer les nucléotides et leur ordre d'enchainement dans un segment d'ADN ou d'ARN donné (Jou et al. 1972). Les analyses qui découlent de la lecture du code génétique sont nombreuses et sont supportées par des logiciels informatiques capables de gérer la grande quantité d'information issue d'un séquençage. Grâce à ces techniques, il est devenu possible d'identifier des anomalies génétiques (Elles & Mountford 2004) ou d'approfondir les connaissances en matière d'évolution (Li et al. 1997). Au cours des dernières décennies, les techniques et technologies ont évolué, notamment en rapidité. Ce qui permet de réduire sensiblement les coûts liés à de telles expériences. La bioinformatique a aussi évolué avec l'afflux de nouvelles données issus de séquençage, où de nombreux défis en algorithmique ont été relevés pour traiter l'information, comme par exemple l'assemblage de contigs (Huang & Madan 1999), l'alignement de séquences (Notredame et al. 1998), la recherche de séquence par alignement local comme *BLAST* (Altschul et al. 1990, Altschul et al. 1997), ou encore la détection de gènes (Burge & Karlin 1997).

2.1.1 Séquençage ancienne génération

Différentes techniques de séquençage ont été inventées à partir de la fin des années 70 comme la dégradation chimique sélective par Maxam et Gilbert en 1977, la synthèse enzymatique sélective par Sanger et Nicklen en 1977 et le pyroséquençage par Ronaghi et al. en 1988.

La méthode de Maxam et Gilbert (Maxam & Gilbert 1977) est basée sur une dégradation chimique de l'ADN, avec une réaction spécifique selon le nucléotide lu. En reconstituant l'ordre des coupures (par dégradation) il est possible de remonter à la séquence. Les inconvénients de cette méthode est l'emploi de produits dangereux, la taille des séquences limitée à environ 250 nucléotides et sa technique complexe à robotiser. La technique se divise en plusieurs étapes : la première consiste à extraire l'ADN, puis de faire un marquage par traceur radioactif. Ensuite il s'opère des clivages selon les bases et l'ADN à séquencer est isolé par électrophorèse puis analysé. La radiographie obtenue est alors analysée informatiquement afin de reconstituer les séquences lues (Mathews & Van Holde 1995).

La méthode de Sanger et Nicklen (Sanger et al. 1977) est basée sur l'interruption de la synthèse enzymatique d'un brin d'ADN complémentaire par l'arrêt de l'élongation. Dans cette approche, l'ADN séquencé est cloné en de nombreuses copies de simples brins et les nucléotides subissent un marquage radioactif. Pour cela, après extraction de l'ADN, il y a amplification par PCR de ce dernier en simples brins. Ensuite, des réactifs sont mis en tube, chacun possédant un type de base marqué radioactivement. Le brin complémentaire est alors polymérisé avec ces bases marquées et les produits de cette réaction sont coulés dans un gel, puis interprétés. Le concept a été amélioré depuis sa découverte et peut séquencer jusqu'à 1200 paires de bases (Lehninger et al. 1993).

Quant à la méthode de Ronaghi (Ronaghi 2001), elle repose sur l'addition d'un nucléotide révélé en temps réel par luminescence. Brièvement, cette méthode débute par la préparation d'un mélange enzymatique nécessaire à la réaction. Ensuite les nucléotides sont ajoutés un après l'autre. Si le nucléotide correspond à celui attendu par la polymérase, il est incorporé et libère un pyrophosphate (PPi) qui est convertit en ATP. Ce dernier fourni l'énergie nécessaire à la conversion d'une luciférine en oxyluciférine qui deviendra luminescente. Les nucléotides

en surplus sont dégradés. Lorsque le signal est capté, la hauteur de l'intensité permet la déduction du nombre de nucléotides répétés. La limite de cette méthode réside dans la limitation de la longueur des brins qui peuvent être séquencés à 500 paires de bases.

2.1.2 Séquençage nouvelle génération (NGS)

Les NGS (*Next Generation Sequencing*) désignent les plateformes actuelles de séquençage qui permettent l'augmentation de la taille des régions séquencées et la réduction drastiques des coûts liés à ces travaux. Elles ont été mises en place par des compagnies de biotechnologies telles que Roche Life Science, Illumina et Applied Biosystem. Bien que toutes ces compagnies font appel à la nanotechnologie et permettent un séquençage de plusieurs centaines de nucléotides à la fois, elles utilisent néanmoins des approches très différentes entre elles. Elles sont le seul moyen actuel pour séquencer des génomes au complet. Cependant, ce séquençage massif s'associe à un compromis sur la qualité des séquences obtenues. Le Tableau 2.1 présente un comparatif des performances des séquenceurs de nouvelle génération actuels et contient des informations telles que la technique biologique utilisée, l'approche d'amplification, le nombre de bases séquençables en une exécution, le coût d'un séquençage par million de bases, etc.

Tableau 2.1 Performances des séquenceurs de nouvelle génération

(Mardis 2008)

	Plateforme		
	Roche (454)	**Illumina**	**SOLID**
Technique	Pyroséquencage	Séquençage par synthèse basée sur une polymérase	Séquençage par ligation
Approche d'amplification	Emulsion PCR	Amplification de pont	Emulsion PCR
Mb par exécution	100	1300	3000
Longueur fragments	250 pb	32-40 pb	35 pb
Cout par Mb	84.39 $	5.97 $	5.81 $

La plateforme Roche 454 repose sur la technique de pyroséquençage et emPCR (PCR en émulsion) capable d'environ 300 000 réactions en parallèle. Elle utilise un support de microplaques (*PicoTiteplates*) qui contient 1,6 millions de puits. Elle est réalisée en plusieurs étapes. Au début, deux adaptateurs sont rajoutés aux extrémités de l'ADN simple brin obtenu par chauffage de l'ADN double brin que l'on souhaite séquencer. Puis, une amplification clonale (PCR) est effectuée. Ce qui permet de démultiplier l'échantillon de départ de manière à éviter les erreurs de séquençage. À la fin de la lecture des brins d'ADN, les séquences sont reconstituées grâce au groupage des contigs.

La technologie de séquençage d'Illumina (CRT Solexa) est une variante de la méthode de Sanger appellée CRT (Cyclic reversible Termination). Elle est capable de séquencer environ 100 000pb à la seconde. Elle utilise des biopuces à ADN et ne comporte pas l'étape lente de PCR ou d'électrophorèse, réduisant ainsi les coûts. Son principe comporte plusieurs étapes. Il est basé sur l'incorporation réversible de nucléotides fluorescents et leur lecture optique. En premier, la banque d'ADN génomique est préparée. L'ADN est ensuite fragmenté, ses extrémités réparées, et des adaptateurs sont fixés sur chaque extrémité. Puis, l'ADN est coulé dans les cellules de la puce et fixé grâce aux adaptateurs. Une formation de ponts d'amplification se produit grâce aux adaptateurs complémentaires présents sur la puce, permettant ainsi de recréer un ADN double brin lors de l'ajout de nucléotides libres fluorescents. Chaque brin créé permet la formation de brins clones lors de l'étape d'amplification. Cette dernière s'effectue en plusieurs cycles. Les fragments sont ensuite lus en utilisant du séquençage par CRT (Cyclic Reversible termination). Cette méthode consiste à lire les nucléotides colorés par fluorescence. Les fragments sont lus sur toute leur longueur, c'est-à-dire 25 à 30 paires de bases, puis une analyse bioinformatique permet de reconstituer les contigs.

Enfin, la technologie SOLID qui signifie "Séquençage par ligation d'oligos et leur détection (*"Sequencing by Oligo Ligation and Detection"*), est capable de séquencer des centaines de millions de petites séquences d'environ 35 paires de bases en un seul processus. Elle est aussi spécialisée dans la détection des substitutions (polymorphismes) et des insertions et délétions (indel). Le procédé adopté par SOLID est l'utilisation de couleurs, 16 couples possibles (nucléotide+couleur) codés par 4 couleurs pour associer les différentes transitions possibles entre deux nucléotides voisins. Chacune des couleurs porte un numéro, de 0 à 3; 0 pour bleu,

1 pour vert, 2 pour orange et 3 pour rouge. Cette technique a un avantage majeur dans l'identification des SNP (Shendure & Ji 2008). Mais ce système pose plusieurs difficultés, notamment lorsqu'une erreur de séquençage survient dans la séquence couleur, où tout le reste de la séquence après l'erreur se retrouve erronée. Une étape d'alignement contre un génome de référence est alors indispensable pour palier ce biais. Aussi, pour faciliter la détection d'erreur de séquençage et permettre la localisation précise de mésappariements éventuels lors de l'alignement, des valeurs de qualité (QV) de séquençage sont attribuées à chaque nucléotide séquencé.

Voici un exemple de nos résultats issu d'un séquençage par SOLID. Cet exemple porte l'identification 1557_1102_758 comme indiqué précédemment. Le format de sortie est en fasta. Ce format contient une première ligne commençant par le symbole ">" suivi d'un nom d'identification, et une seconde ligne (ou multilignes) contenant de l'information. Habituellement, l'information est une séquence génétique en nucléotide, mais ici elle est donnée sous forme de séquence couleur et de qualité.

Séquence couleur dans un fichier csfasta :

```
>1557_1102_758_F3
T032022313123021010103233020103031 31
```

Qualité dans un fichier qual :

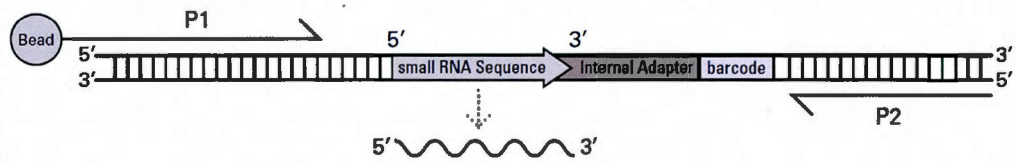
```
>1557_1102_758_F3
22 22 26 11 24 26 27 29 20 28 31 29 25 20 29 24 28 28 14 31 29
28 28 24 29 20 28 28 24 28 26 22 26 26 31
```

Une séquence de bonne ou mauvaise qualité se défini par la valeur des qualités attribuées aux premières nucléotides séquencées. Les chiffres varient selon les études. Cependant, il est largement accepté que si dans les 10 premiers nucléotides nous avons une valeur de qualité inférieure à 10, alors le fragment couleur peut être considéré de mauvaise qualité. Voici un exemple de fragment de mauvaise qualité, à la différence de celui présenté pour le fragment 1557_1102_758_F3 :

```
>1_99_650_F3
9 6 3 3 2 4 8 2 8 5 14 3 4 6 8 5 4 5 7 10 5 6 4 6 12 12 9 5 9 3 8 9 6 11 16
```

La technique du séquençage par SOLiD™ repose sur trois grandes étapes : La première est la génération des librairies par l'utilisation du kit Small RNA Expression Kit (SREK), basée sur l'attachement de deux adaptateurs autour de la séquence d'ADN ou d'ARN, dont un universel dénommé P1 et un autre plus complexe dit P2. L'organisation des adaptateurs et d'une séquence d'ARN est schématisée dans la Figure 2.1A. L'adaptateur P2 est séparé en 3 parties : un adaptateur interne, un code barre spécifique à une librairie sous forme de nucléotides (Figure 2.1B), et une séquence universelle qui marque la fin de l'adaptateur.

A



B

Component	3' primer sequence (shown 5' to 3')
SOLiD PCR Primers 1	CTGCCCCGGGTTCCTCATTCTCT***AAGCCC***CTGCTGTACGGCCAAGGCG
SOLiD PCR Primers 2	CTGCCCCGGGTTCCTCATTCTCT***CACACC***CTGCTGTACGGCCAAGGCG
SOLiD PCR Primers 3	CTGCCCCGGGTTCCTCATTCTCT***CCCCTT***CTGCTGTACGGCCAAGGCG
SOLiD PCR Primers 4	CTGCCCCGGGTTCCTCATTCTCT***CATCGG***CTGCTGTACGGCCAAGGCG
SOLiD PCR Primers 5	CTGCCCCGGGTTCCTCATTCTCT***TCGTTG***CTGCTGTACGGCCAAGGCG
SOLiD PCR Primers 6	CTGCCCCGGGTTCCTCATTCTCT***GGGCAC***CTGCTGTACGGCCAAGGCG
SOLiD PCR Primers 7	CTGCCCCGGGTTCCTCATTCTCT***CCAGAC***CTGCTGTACGGCCAAGGCG
SOLiD PCR Primers 8	CTGCCCCGGGTTCCTCATTCTCT***CTCCGT***CTGCTGTACGGCCAAGGCG
SOLiD PCR Primers 9	CTGCCCCGGGTTCCTCATTCTCT***CCCTTC***CTGCTGTACGGCCAAGGCG
SOLiD PCR Primers 10	CTGCCCCGGGTTCCTCATTCTCT***GCGGTC***CTGCTGTACGGCCAAGGCG

P1: CCACTACGCCTCCGCTTTCCTCTCTATGGGCAGTCGGTGAT

Figure 2.1 Organisation des adaptateurs SOLID4

(SOLID smallRNA Expression Kit)

Deuxièmement, l'ADN est amplifié par emPCR et attaché à des microbilles. Enfin, toutes les billes sont étalées sur une plaque et le séquençage débute. La Figure 2.2 illustre ces étapes. Dans la partie *a*, processus de séquençage, un primer universel s'attache à l'adaptateur P1, lui-même fixé à une bille. Des oligos s'attacheront sur la séquence à séquencer après le primer P1 en 5' grâce à une ligase. Les oligos contiennent des jeux de séquences de deux

nucléotides qui émettent une couleur précise. Le séquençage peut être effectué plusieurs fois (*round*) afin d'obtenir les SNP ou détecter les erreurs de séquençage décrites dans la partie *b*.

Pour ce projet de maîtrise, le choix d'employer la plateforme SOLID était d'ordre matériel d'une part (disponibilité d'une plateforme à l'Université de Montréal), de coût et de facilité d'accès d'autre part.

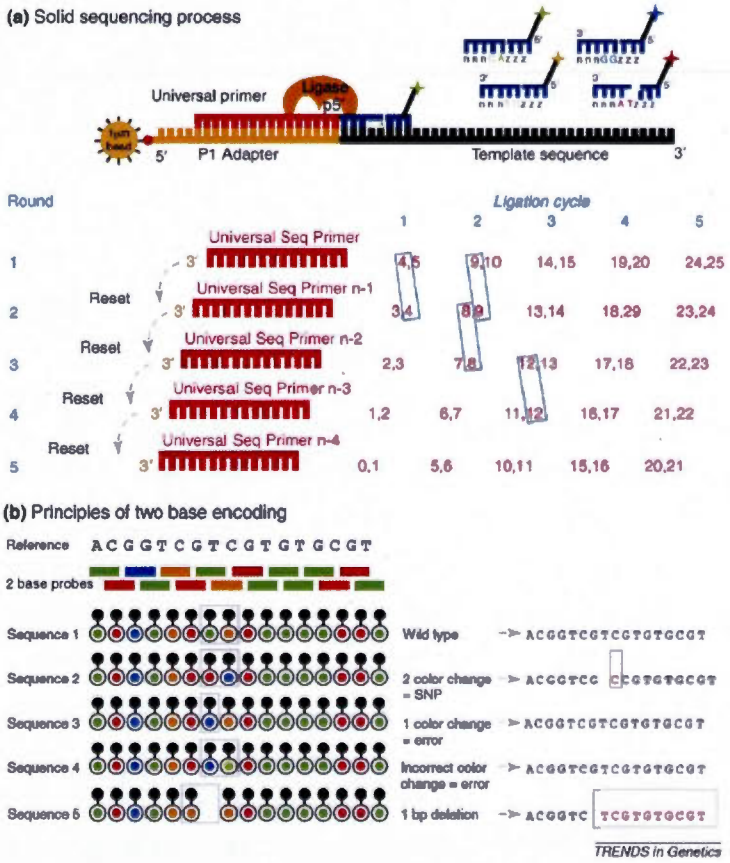


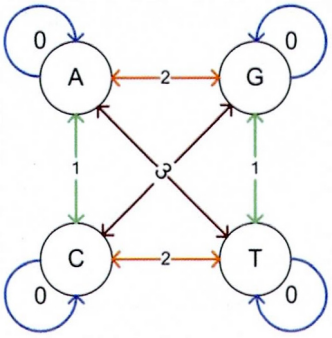
Figure 2.2 Étapes du séquençage d'ABI SOLID

(SREK: SOLiD™ Small RNA Expression Kit)

2.2 Traitement des séquences couleurs

Les résultats issus de la plateforme de séquençage AB SOLID ne peuvent pas être manipulés directement en tant que séquence lors de l'alignement. Car elle repose sur une méthode de

séquençage en double base, avec des couleurs qui permettent une transition d'une base à une autre (comme présenté précédemment). Ce ne sont donc pas des nucléotides qui sont fournis par la plateforme, mais des transitions chiffrées. Les différentes transitions sont présentées à la Figure 2.3a sous forme d'automate à état fini accouplé à un processus markovien caché. La Figure 2.3b montre la transformation d'une séquence nucléotidique en séquence couleur.



Séquence en code couleur
T02111210023102
| | | | | | | | | | | | | |
TCACAGTTTCGTT
Séquence nucléotidique

A B

Figure 2.3 Transitions des codes couleur sur la plateforme ABI SOLID

(Rumble et al. 2009)

Avant d'aligner, il est très conseillé d'enlever auparavant le restant d'adaptateur interne (voir Figure 2.1A). En effet, pour des raisons de coûts, le séquençage mélange toutes les librairies pour ne faire qu'une seule exécution. Lorsque le séquençage est terminé, les séquences sont réorganisées en librairies grâce aux codes-barres, puis coupées à la longueur maximale désirée qui est de 35 nucléotides. Toutes les séquences après le séquençage sont de cette longueur, car elles contiennent les séquences réelles, et le début de l'adaptateur interne. Si une séquence fait 35pb, alors il n'y aura plus de trace de l'adaptateur. Eliminer l'adaptateur restant peut s'effectuer avec des programmes de recherche de motifs comme *cutadapt* (Schulte et al. 2010). Ce dernier a un avantage associé à sa prise en charge des erreurs de séquençage. La séquence couleur 1557_1102_758 sans son adaptateur s'obtient de la manière suivante, où les codes rouges ne sont pas pris en charge lors de la coupure :

Séquence de l'adaptateur 330201030313112312
Séquence couleur T032022313123021010103233020103031 31
Séquence sans l'adaptateur 32022313123021010103

2.3 Alignement de fragments de séquences sur un génome de référence

L'alignement (*mapping* en anglais) permet de localiser des millions ou milliards de courtes séquences obtenues par séquençage haut débit, appelées fragments (*reads* en anglais), dans d'autres séquences connues, souvent des génomes de référence ou des ESTs. Il permet par la même occasion de convertir correctement les séquences couleurs en séquences nucléotidiques. Il est important de distinguer l'alignement de l'assemblage, car ce dernier permet de construire des régions longues à partir de plusieurs morceaux séquencés, et le premier permet seulement la localisation la position des séquences sur une séquence de référence connue.

Pour cela, nous pourrions utiliser l'algorithme le plus connu, *BLAST*, qui effectue des alignements locaux. Mais il se révèle lent et nécessite beaucoup d'espace mémoire compte tenu de la taille des données. Des logiciels plus spécialisés, capables d'effectuer l'alignement sur une grande quantité de séquences et qui prennent en compte des mésappariements possibles avec la référence, ont alors été développés. Ces derniers emploient des techniques d'indexage et de compression avancées des séquences. Ceci permet d'optimiser l'usage de la mémoire occupée pour ne pas la saturer et d'augmenter la vitesse de calcul. D'autre part, l'ADN possède naturellement de longues régions qui se répètent, et il devient difficile de localiser précisément ces séquences. Ce même problème existe aussi lors de l'assemblage d'un génome complet, car la faible taille des morceaux séquencés ne permet pas de reconstruire correctement l'ordre dans lequel les séquences sont placées. Dans ce cas, les programmes d'alignement peuvent soit renvoyer plusieurs résultats possibles, ou sélectionner la meilleure position en se basant sur la qualité de séquençage de la séquence à aligner. Il faut noter que des défis supplémentaires associés à la qualité du séquençage sont à prévoir, telles que les erreurs de séquençage, l'alignement en tenant compte des introns, la faculté de travailler avec de petites séquences et la présence de SNP (Trapnell & Salzberg 2009). Par exemple, concernant les introns, l'alignement s'effectue avec des fragments provenant potentiellement de régions épissées lors d'un séquençage d'ARN. Les fragments qui proviennent de régions d'intersection entre deux exons ont besoin d'être alignés différemment de ceux qui sont complètement inclus dans un exon. Pour cela, après avoir

aligné les fragments sur le génome, ceux qui n'ont pu l'être sont soumis à *TopHat* (Trapnell et al. 2009) ou *ERANGE* (Mortazavi et al. 2008). Ces programmes pourront aligner les fragments sur des exons séparés par des introns. Puisque nous utilisons des ESTs en tant que référence, ce problème des introns n'est pas rencontré dans notre étude.

Ainsi, les défis à relever pour aligner des millions de petites séquences sur des génomes sont la vitesse de calcul, la gestion des séquences couleurs, la gestion de la mémoire et la répétition des courtes séquences quasi-similaires. Pour arriver à cette fin, depuis quelques années, plusieurs logiciels ont vu le jour, tels que *MAQ* (Li, Ruan & Durbin 2008), *Bowtie*, *SHRiMP* (Rumble et al. 2009), *SOAP* (Li, Li, Kristiansen), *BFAST* (Homer et al. 2009), *rna2map* (ABI SOLID), *Bioscope* (ABI SOLID) et d'autres encore que l'on peut retrouver dans le Tableau 2.2.

Tableau 2.2 Caractéristiques de quelques logiciels d'alignement

(Trapnell & Salzberg 2009)

Programme	Site web	Source libre ?	Prend en charge couleurs AB SOLID	Longueur maximale fragments
Bowtie	bowtie.cbcb.umd.edu	Oui	Non	Aucune
BWA	maq.sourceforge.net/bwa-man.shtml	Oui	Oui	Aucune
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Mosaik	bioinformatics.bc.edu/marthlab/mosaik	Non	Oui	Aucune
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
SOAP2	soap.genomics.org.cn	Non	Non	60
ZOOM	bioinfor.com	Non	Oui	240

Chacun possède une approche quelque peu différente sur la gestion des données et de la mémoire, avec des algorithmes optimisés pour des plateformes de calcul parallèles ou de simples ordinateurs. Dans les sections suivantes, nous présentons plus de détails sur les méthodes utilisées durant cette maîtrise.

2.3.1 Méthodes d'alignement MAQ et Bowtie

Les programmes tels que *MAQ* et *Bowtie* utilisent une stratégie d'indexage du génome de référence qui permet d'accélérer les étapes d'alignement des fragments contre le génome de référence. Comme on peut le voir dans la Figure 2.4a, *MAQ* possède un algorithme basé sur l'indexage de germes (seed), où chaque position de la référence est coupée en pièces égales puis stockées dans une table de correspondance. Chaque fragment est aussi coupé de la même manière et est comparé au génome de référence en appelant les clés de la table.

D'un autre point de vue, comme présenté dans la Figure 2.4b, l'algorithme de *Bowtie*, plus rapide que celui de *MAQ*, est basé sur la transformation Burrows-Wheeler. Dans lequel le génome de référence est stocké efficacement en mémoire par concaténation et indexage. Les fragments sont alignés caractère par caractère contre le génome et l'algorithme met à jour à chaque fois l'intervalle (indiqué en faisceau bleu sur la figure) dans lequel sont retrouvés les nucléotides. Si le fragment ne s'aligne pas parfaitement sur la référence, l'algorithme revient en arrière, change le dernier nucléotide et recherche de nouveau l'alignement.

Le défaut de *Bowtie* par rapport à *MAQ* est qu'il ne prend toutefois pas en charge les séquences couleurs générées par la plateforme de séquençage AB SOLID.

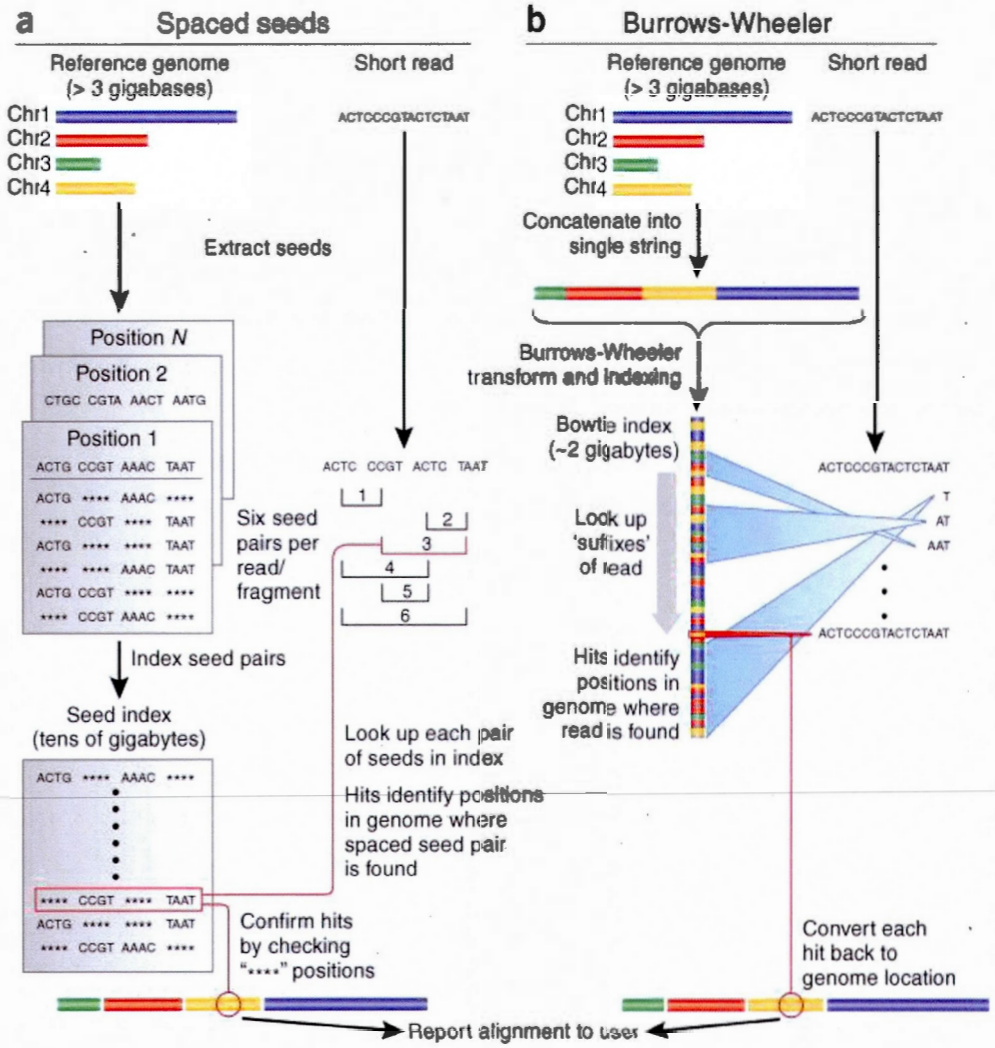


Figure 2.4 Algorithmes de *MAQ* (Spaced seeds) et *Bowtie* (Burrows-Wheeler)

(Trapnell & Salzberg 2009)

La complexité temporelle de *MAQ* est de l'ordre de $O(N \log N)$, et plus précisément $O(N \log N + L + 2^{-k}NL)$, où N représente le nombre de fragments à aligner sur une référence de longueur L en utilisant k bits d'indexage (24 pour *MAQ*). Le premier terme $N \log N$ correspond au temps passé pour ordonner les index, le second, L, à la lecture de la séquence de référence, et le troisième $2^{-k}NL$ à l'alignement. Pour Bowtie, sa complexité temporelle est de l'ordre de $O(N^2 \log N + l \times \frac{L}{4^k})$, où l est la longueur des fragments. Comme

précédemment pour *MAQ*, la formule se divise en deux termes, l'un construit l'index et l'autre sert à l'alignement.

MAQ nécessite un format particulier en entrée, appelé *fastq*. Dans le cas des données SOLID, il possède un outil qui permet de prendre en compte les fichiers de séquences couleur ainsi que leur fichier de qualité. Le programme *Cutadapt* (voir section 2.2) procède aussi de la même manière. *MAQ* et *Cutadapt* combinent les deux fichiers *csfasta* et *qual* en un seul fichier qui porte l'extension *fastq*. Ce dernier contient les résultats présentés comme celui-ci :

```
@1557_1102_758/1
TGAGGTCTCGTAGCACACAT
+
7;,9;<>5=@>:5>9==/@>
```

La ligne `TGAGGTCTCGTAGCACACAT` est une fausse séquence, car elle est encore en couleur, où A=0, C=1, G=2 et T=3. La ligne `7;,9;<>5=@>:5>9==/@>` regroupe les valeurs de qualité pour chaque nucléotides. Elles sont obtenues simplement en faisant le lien entre les valeurs de la table ASCII lues sur le fichier de qualité en y rajoutant 33 et le code numérique des caractères correspondants. Le chiffre 33 est additionné car dans table ASCII les caractères visibles commencent à cette valeur. Une fois ce format obtenu, *MAQ* peut alors faire son alignement.

Lorsque l'alignement est effectué, quatre informations importantes peuvent être soutirées des résultats bruts obtenus. Voici l'information obtenue pour le fragment d'exemple :

```
1557_1102_758/1 gi|143479898|gb|CJ850028.1|CJ850028     105     +       0       00
0       0       0       0       69      0       20      tgaggTCtCGTagCaCAcAT
7;,9;<>5=@>:5>9==/@>
```

La séquence couleur 1557_1102_758 a pu être alignée sur l'EST portant l'accession gi|143479898|gb|CJ850028.1|CJ850028, à la position 105 dans le sens positif (+) avec 0 mésappariements.

À partir de là, nous pouvons extraire la séquence du petit ARN, ainsi celle qui contiendrait le précurseur sur l'EST gi|143479898|gb|CJ850028.1|CJ850028 à la position 105. La façon d'extraire cette dernière varie selon les sources. Suivant la position théorique d'un microARN sur l'épingle, nous avons choisi de récupérer -160pb avant la position de départ de

l'alignement sur le génome, et 20pb après la position de fin de l'alignement. Sur le bras gauche, c'est l'inverse. Nous obtenons donc un petit ARN sous sa forme nucléotidique, TAGGAGCATGATTCAACCAA, le nom de l'EST est indisponible puisque non annoté, et deux grandes séquences qui contiennent le futur précurseur :

```
Séquence a: TAGGAGCATGATTCAACCAA     Unknown
TTGCTCAATGAGATCGTGAATAGGAGCATGATTCAACCAATTGACATCAATGTTGATAAAGGCATGGA
AAAATCTTATCGTATACATGATATGGTGATTGATTTCATATGCTGAAGAAAAACACTTTTGTGGCATG
GAGTGAAAGCGGTCTATCCAAAGGCAAGATGTGAAGATTATCCATCTGAAAGAGCATGACAGGT

Séquence b: TAGGAGCATGATTCAACCAA     Unknown
AGATTAAGCAAGCTCATTCGATATGTAAGGTGGATAGTTTGAGGTTTTTGTTTATCATGAAAAACAAG
GGATTAGCAAGTTTGAGCTCGGTGGTAGAATTGCTCAATGAGATCGTGAATAGGAGCATGATTCAACC
AATTGACATCAATGTTGATAAA
```

2.3.2 SHRiMP

SHRiMP (SHort Read Mapping Package) (Rumble et al. 2009), est un jeu d'algorithmes améliorés des précédentes méthodes d'alignements capable de gérer la présence de polymorphismes. Cette suite permet un alignement des séquences couleurs directement sur un génome. Cependant, elle ne traite par défaut que les séquences dont la longueur est supérieure à 24pb. Car en dessous de cette limite, l'algorithme produit de nombreux faux positifs. Un paramétrage complexe des germes est nécessaire pour diminuer cette limite.

L'algorithme de *SHRiMP* exploite trois techniques d'alignement de séquences menant à la localisation des fragments (Figure 2.5): les approches de filtre Q-gram (Rasmussen et al. 2006), les *space seeds* (Califano & Rigoutsos 1993), et l'algorithme de Smith-Waterman accéléré par vectorisation (Farrar 2007, Rognes & Seeberg 2000, Wozniak 1997).

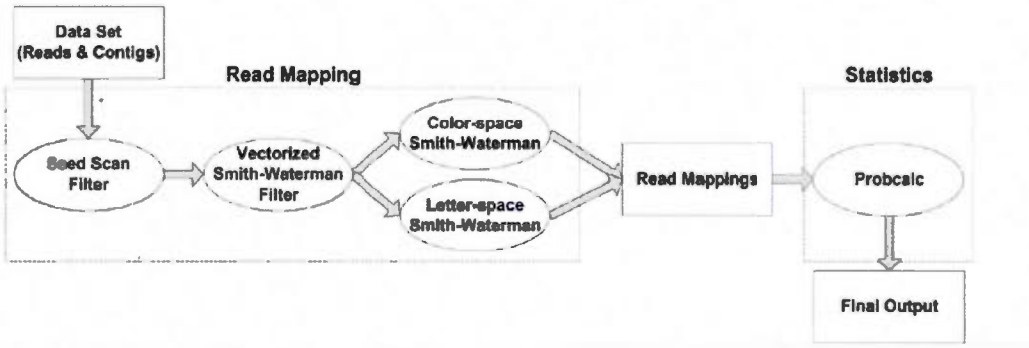


Figure 2.5 Étapes de l'algorithme de *SHRiMP*
(Rumble et al. 2009)

Spaced Seeds : La plupart des méthodes heuristiques d'alignement local reposent sur l'identification de germes, qui sont de petits alignements parfaits (correspondance exacte et aucun mésappariement) entre deux séquences. L'avantage de trouver des correspondances exactes permet l'utilisation de méthodes d'indexation à temps quasi constant comme des tables de hachage, tables de suffixes etc. La particularité des *spaced seeds* est la possibilité d'avoir des correspondances non contigües le long de la séquence à analyser. Ils ne sont constitués que de 1 et 0 et possèdent deux caractéristiques : une longueur, déterminée par le nombre de caractères du germe, et un poids, obtenu en comptant la quantité de 1. Ainsi, le germe 1111100011111010l a une longueur de 17 et un poids de 12. Ils peuvent aussi inclure des positions prédéterminées de mésappariement, indiquées par les 0. Les germes très courts ont tendance à être aligné trop souvent. Il faut donc considérer plusieurs germes dans une même région pour qu'elle soit prise en compte par le programme d'alignement.

Filtres Q-gram : Alors que la plupart des programmes d'alignement, tel que *BLAST*, utilisent un germe de départ pour entamer un alignement en acceptant des caractères désappariés autour, les filtres Q-gram nécessitent plusieurs germes pour déterminer si une correspondance correcte existe.

Smith-Waterman (Smith & Waterman 1981) est un algorithme de programmation dynamique qui permet d'effectuer des alignements locaux de séquences. Il compare les segments de toutes les longueurs possibles en optimisant la mesure de similarité. Il se défini par une matrice H d'entiers construite à partir de deux séquences à aligner :

$$H(i,0) = 0, 0 \leq i \leq m \qquad (1)$$

$$H(0,j) = 0, 0 \leq j \leq n \qquad (2)$$

Si $a_i = b_j, w(a_i b_j) = w$ (appariement) ou si $a_i \neq b_j, w(a_i b_j) = w$ (mésappariement)

$$H(i,j) = \max \begin{Bmatrix} 0 \\ H(i-1,j-1) + w(a_i, b_j) \text{ Match/Mismatch} \\ H(i-1,j) + w(a_i, -) \text{ Deletion} \\ H(i,j-1) + w(-, b_j) \text{Insertion} \end{Bmatrix}, 1 \leq i \leq m, 1 \leq j \leq n \qquad (3)$$

où a et b sont des séquences; i et j les positions sur ces séquences; m la longueur de a; n la longueur de b; H*(i,j)* le score maximum de similarité entre un suffixe de $a[1..i]$, w le score (poids) et un suffixe de $b[1..j]$ et – les gaps.

2.3.3 Comparaison et limites des programmes d'alignement

Les méthodes d'alignement sont limitées à plusieurs niveaux, comme l'absence d'outils standards pour aligner les fragments encodés sous différents formats (e.g. exploitation de séquences couleurs) et la variation sur la taille des fragments permise par les différents programmes. Une étude comparative sur l'efficacité de l'alignement de différents programmes menée par Rumble *et al.* en 2009 a montré des résultats mitigés. Le Tableau 2.3 montre les résultats d'alignement de 135 millions de fragments SOLID de 35pb provenant de l'espèce *Ciona savignyi* à partir du programme *SHRiMP* et celui d'ABI SOLID. Dans ce tableau, le pourcentage d'alignement pour les fragments uniques est d'environ 38% avec *SHRiMP* et 11% avec le programme d'ABI SOLID.

Tableau 2.3 Comparaison de l'alignement entre *SHRiMP* et ABI SOLID

(Rumble et al. 2009)

	SHRIMP	**SOLID Mapper**
Fragments alignés plusieurs fois	64 252 692 (47.7%)	12 602 387 (9.4%)
Couverture moyenne	10.3	3.0
Délétions	51 592	0
Insertions	19 970	0

Dans le Tableau 2.4, plusieurs autres programmes sont comparés au niveau des temps de calcul et du pourcentage d'alignement sur des jeux de données réels provenant de deux plateformes de séquençage (Illumina et SOLID) (Rumble et al. 2009). Les résultats montrent clairement aussi des disparités entre les programmes d'alignement. Par exemple, les temps de calculs pour aligner un million de fragments ABI SOLID de 25pb varient entre 2977 et 21179 secondes pour un alignement de 2.4% et 74.7% par *SHRiMP* et *BWA* respectivement. Dans ce cas là, *SHRiMP* démontre son incapacité à aligner des courtes séquences ABI SOLID, mais est l'un des meilleurs pour aligner les longs fragments provenant des deux plateformes de séquençage. *MAQ* et *BFAST* sont de bons compromis, où leurs pourcentages d'alignement sont parmi les plus élevés dans tous les cas, et ce pour des temps de calcul très raisonnables par rapport aux autres programmes. Ainsi, chacun d'entre eux a des forces et des faiblesses, mais il faut souligner que le pourcentage d'alignement n'indique pas si ce sont les mêmes fragments qui ont pu être alignés.

Tableau 2.4 Temps de calcul et pourcentage de séquences alignées par différents programmes

(Rumble et al. 2009)

	Illumina 10.9 M 36 bp reads	Illumina 10.9 M 36 bp reads	Illumina 3.5 M 55 bp reads	Illumina 3.5 M 55 bp reads	ABI SOLiD 1 M 25 bp read	ABI SOLiD 1 M 25 bp read	ABI SOLiD 1 M 50 bp read	ABI SOLiD 1 M 50 bp read
	Time (s)	% mapped	Time (s)	% mapped	Time (s)	% mapped	Time (s)	% mapped
BFAST	43,775	32.1	47,474	69.6	9,590	66	42,856	72.5
BLAT*	68,758	24.3	6,735,069	77.4	NA	NA	NA	NA
Bowtie	2,270	13.1	857	55.7	NA	NA	NA	NA
BWA	7,682	16	4,883	59.3	21,179	74.7	845	47.8
MAQ	8,607	28.7	126,541	73.6	7,602	63.6	6,680	68.1
SHRiMP*	186,764	14.9	324,380	83.3	2,977	24	32,644	70.4
SOAP	11,938	13.3	131,248	62.4	NA	NA	NA	NA

Bien qu'il existe plusieurs méthodes qui ont des résultats acceptables sur les données de NGS, toutes celles-ci ont des résultats mitigés pour aligner les courtes séquences. Cela est dû fondamentalement à la présence de grandes portions de séquences identiques dans un génome de référence qui ne peuvent être toutes supportées par les algorithmes d'optimisation. En effet plus la séquence recherchée est petite, plus la probabilité d'en retrouver une similaire par hasard est grande. Enfin, si le génome de référence est incomplet, par exemple seulement constitué d'ESTs, comme celui du blé, le pourcentage d'alignement peut être affecté par la partialité du génome, l'absence des régions introniques et la qualité du séquençage des ESTs.

2.4 Prédiction bioinformatique des microARNs

Le problème de la prédiction bioinformatique des microARNs peut être formalisé de la façon suivante : Étant donné une séquence *S*, déterminer si *S* est un précurseur. Pour cela, les méthodes actuelles tiennent compte de plusieurs caractéristiques spécifiques des précurseurs, dont la taille, la structure secondaire, l'énergie libre, la composition en bases azotées, le taux de GC, et leur taux de conservation dans les autres espèces. Notons que la structure secondaire est l'une des caractéristiques principales exploitées par les algorithmes de prédiction. Elle est préalablement prédite par des algorithmes de programmation dynamique, d'analyse de la covariation, de minimisation de l'entropie, et par la recherche de patrons de séquences complémentaires (Dirks et al. 2004). En plus de la forme de l'épingle à cheveux caractéristique d'un précurseur qui porte un microARN, d'autres caractéristiques sont prises en compte dans la prédiction. Telles que le nombre et la position de la(des) boucle(s), le nombre de bases dans la boucle, le nombre de bases pairées par complémentarité entre les branches de l'épingle, et le nombre de bases inclues dans des hernies. L'information du précurseur n'est pas suffisante pour distinguer correctement les vrais précurseurs. Pour cela, il faut ajouter les informations issues du séquençage. L'évidence d'une séquence exprimée et son abondance sont des éléments essentiels pour la réduction des faux positifs. Dans ce cas présent, il serait intéressant d'analyser la position du microARN sur un précurseur prédit afin de déterminer s'il est correctement placé. Dans le chapitre III, nous présentons notre approche de validation de la position et l'abondance des microARNs associés à un précurseur.

Il est important de noter que plusieurs milliers de microARNs et leurs précurseurs ont été identifiés expérimentalement. Ils sont stockés dans les bases de données telles que miRbase et PMRD. En parcourant tous les microARNs publiés et validés, il est possible de modéliser les paramètres communs dans une méthode d'apprentissage machine et de créer des tests positifs ou des jeux d'entrainement pour les logiciels de prédiction.

Les sections qui suivent décrivent en premier, l'une des techniques permettant de prédire la structure secondaire d'un ARN et en second, deux logiciels dérivés d'algorithmes d'apprentissage machine qui permettent de prédire si une séquence, rattachée à sa structure secondaire, est un précurseur valide ou simplement une partie du génome pouvant se replier dans la même forme que les vrais précurseurs.

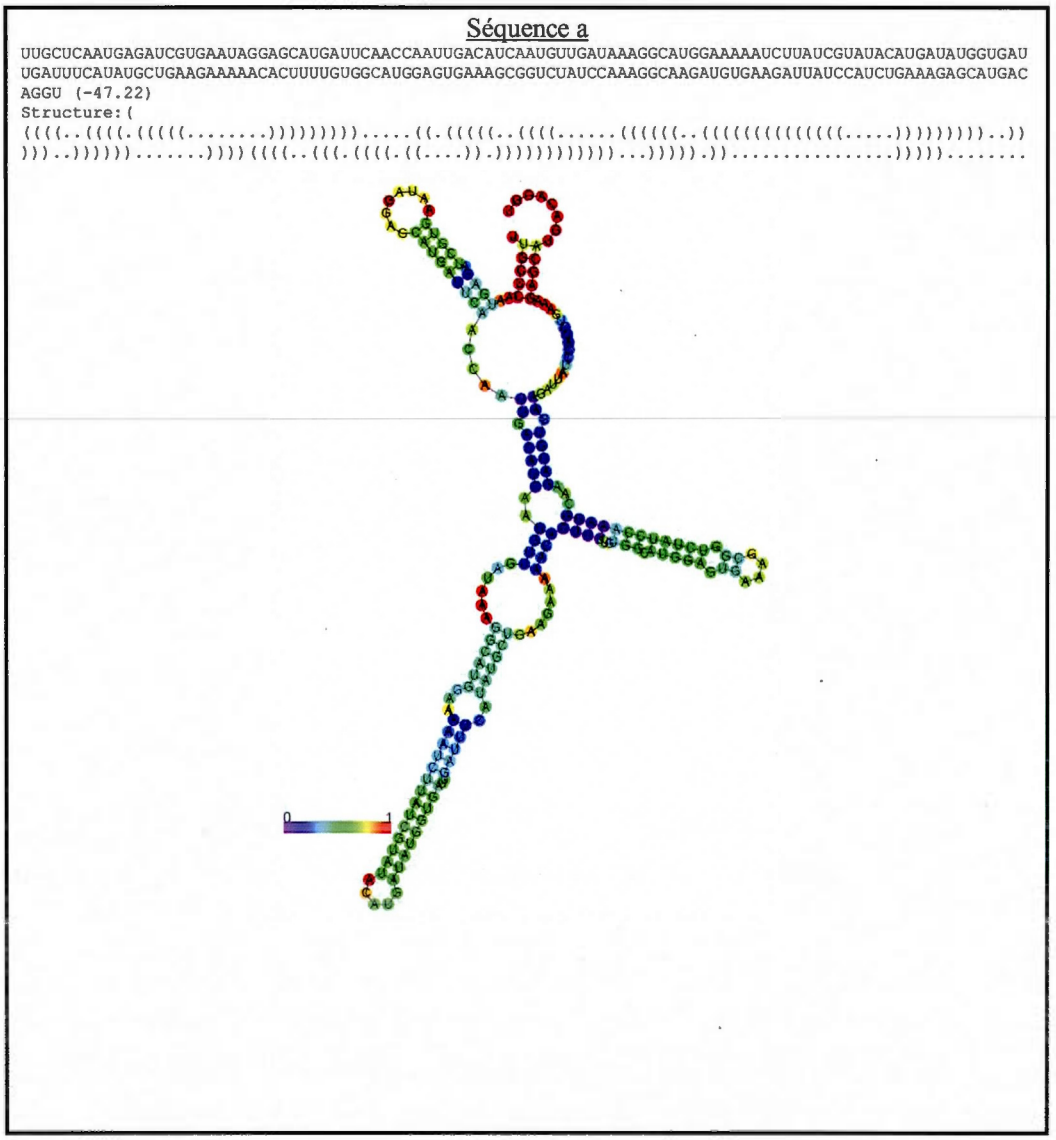
2.4.1 Prédiction de structures secondaires des ARNs

La prédiction de structures secondaires d'ARN est un vaste champ d'étude qui ne se limite pas au simple cadre des microARNs. Elle concerne en effet toutes les recherches liées à l'ARN, puisque c'est la conformation en structure non linéaire qui lui confère une fonction particulière. Ceci s'observe autant pour les pré-miRNAs que pour les ARNs de transfert, les ribozymes, ou encore les ARNs ribosomaux. Dans la littérature il existe plusieurs approches de prédiction de structures secondaires. La plus ancienne est MFold qui date de 1981 (Zuker & Stiegler 1981), et s'en sont suivi différentes optimisations retrouvés dans *RNAfold* (Hofacker et al. 1994) et MC-SYM (Major et al. 1991). *RNAfold*, que nous utilisons dans ce mémoire, est un des outils les plus employés dans les études de prédiction des précurseurs (Unver et al. 2009) (Leung et al. 2007). Il est facile à utiliser et il est implémenté pour pouvoir efficacement replier la structure d'ARN contenant plusieurs structures secondaires circulaires ou en boucle au contraire de MFold qui contient des optimisations pour des repliements linéaires (Hofacker & Stadler 2006). Par ailleurs, il existe des méthodes basées sur l'analyse de covariations comme COVE (Eddy & Durbin 1994), Vienna (Hofacker 2004) ou sur des méthodes probabilistes comme QRNA (Rivas & Eddy 2001). Pour tous les algorithmes, il existe plusieurs repliements qui possèdent une structure optimale. Ces structures sont basées sur les caractéristiques de l'énergie libre tels que :

- L'énergie libre d'une sous structure est négative si et seulement si c'est une paire d'appariements
- Un appariement G-C est plus stable qu'un appariement A-U
- Les zones externes non appariées ont un score nul
- Si l'énergie libre totale d'une structure est strictement positive, alors cette structure ne peut pas être stable
- Une structure est optimale si toutes ses sous structures sont optimales
- Une paire G-U ne déstabilise pas la structure mais possède un score de 0
- L'énergie associée à une paire de base est influencée seulement par la paire précédente

L'algorithme qui permet de calculer les paires de bases d'une structure secondaire est basé sur les techniques de programmation dynamique. Il permet d'éviter de recalculer les structures optimales de sous-structures déjà calculées.

Dans nos données d'exemple de la séquence couleur 1557_1102_758, les deux séquences extraites sur l'EST ont leurs nucléotides T remplacés par des U par *RNAfold* et donnent les repliements présentés en Figure 2.6. Les couleurs renseignent sur le score de probabilité de pairage entre les bases.



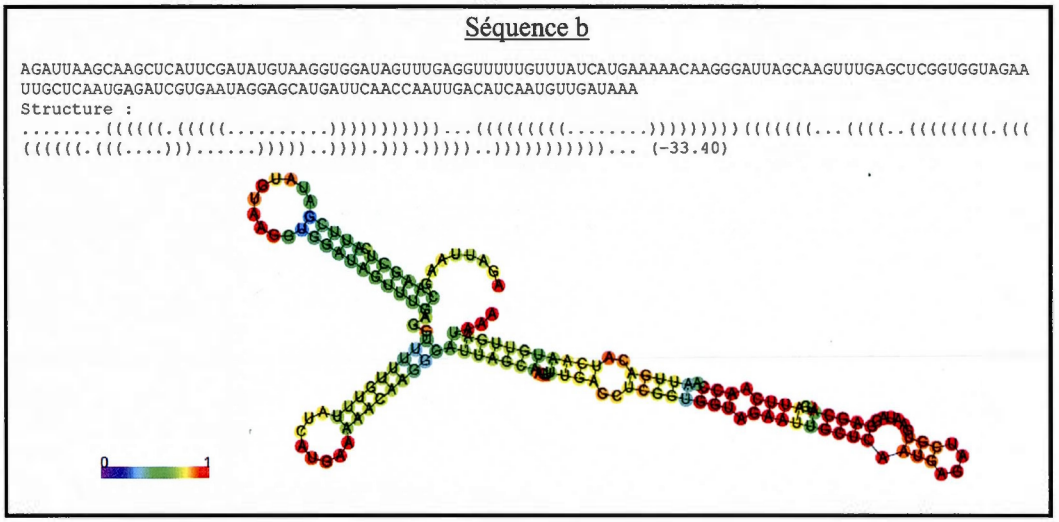


Figure 2.6 Exemples de structures secondaires des précurseurs potentiels

2.4.2 Outils de prédiction des précurseurs de microARNS

Les structures secondaires prédites sont soumises à des logiciels de prédiction de précurseurs de microARNs afin de vérifier si elles contiennent des précurseurs potentiels. À cette fin, plusieurs programmes existent. Parmi les plus récents, le programme *Novomir* (Teune & Steger 2010) utilise une série de filtrages et un modèle de Markov caché dérivé des données de miRbase. La particularité de ce logiciel par rapport aux autres concerne sa capacité à prédire plusieurs positions possibles du microARN sur le précurseur. Par ailleurs, il existe des méthodes adaptées à des espèces. Par exemple *MIRcheck* pour les plantes (Jones-Rhoades 2010), *MIRseeker* (Lai et al. 2003) pour les insectes, et *Vir-Mir db* (Li, Shiau & Lin) pour les virus. D'autres méthodes sont plus générales comme *MIRfinder* (Huang et al. 2007), *miRAlign* (Wang, Zhang, Li,), *MirEval* (Ritchie et al. 2008), *miRank* (Xu et al. 2008), *ProMir* (Nam et al. 2006), *Triplet-SVM* (Xue et al. 2005), *miRdeep* (Friedlander et al. 2008, Friedlander et al. 2012), *miRanalyzer* (Hackenberg et al. 2011, Hackenberg et al. 2009) etc., de nouveaux étant publiés régulièrement chaque année. Seuls miRdeep et miRanalyzer prennent en charge des données de séquençage haut débit.

Deux logiciels utilisés dans cette étude sont décrits ici : *miPred* (Jiang et al. 2007) et *HHMMiR* (Kadri et al. 2009). Le premier est basé sur l'organisation de triplets de nucléotides

et des arbres de décision, le second exploite les modèles probabilistes basés sur les modèles de Markov cachés.

2.4.2.1 *MiPred*

MiPred (Jiang et al. 2007) est un autre algorithme de prédiction de microARNs qui utilise l'apprentissage machine pour distinguer les vrais précurseurs d'autres séquences en épingles. Le programme se base sur la composition de triplets de la structure, l'énergie libre minimale (MFE), la p-value de tests de permutation aléatoires, et sur des modèles de forêt d'arbres décisionnels (Random Forest Prediction Models). D'un point de vue technique, *MiPred* est codé en *R* et nécessite en entrée une séquence et sa structure secondaire prédite par un programme de repliement, et passe par une étape de sélection des épingles présentes sur la structure. *MiPred* emploie comme référence des précurseurs réels et publiés comme données d'entrainement, ainsi que sur des pseudos précurseurs qui ressemblent à des épingles mais qui ne portent pas de microARN.

À partir de la structure secondaire et de la séquence, à l'image d'un autre logiciel, *Triplet-SVM*, *miPred* extrait des triplets contenant un nucléotide de départ (A, U, G ou C), et trois sous structures adjacentes composées de parenthèses ouvrantes ou fermantes ou de points. Il y a 32 triplets possibles, caractérisant un vecteur de paramètres. Ils sont comptés le long de la séquence et de la structure puis normalisés (voir Figure 2.7).

Un score général donné par *MiPred* repose sur l'analyse de la composition des triplets de la structure et leur degré d'apparition dans des données d'entrainement, ainsi que sur la probabilité que cette structure puisse être trouvé de façon aléatoire et la MFE (voir Tableau 2.5).

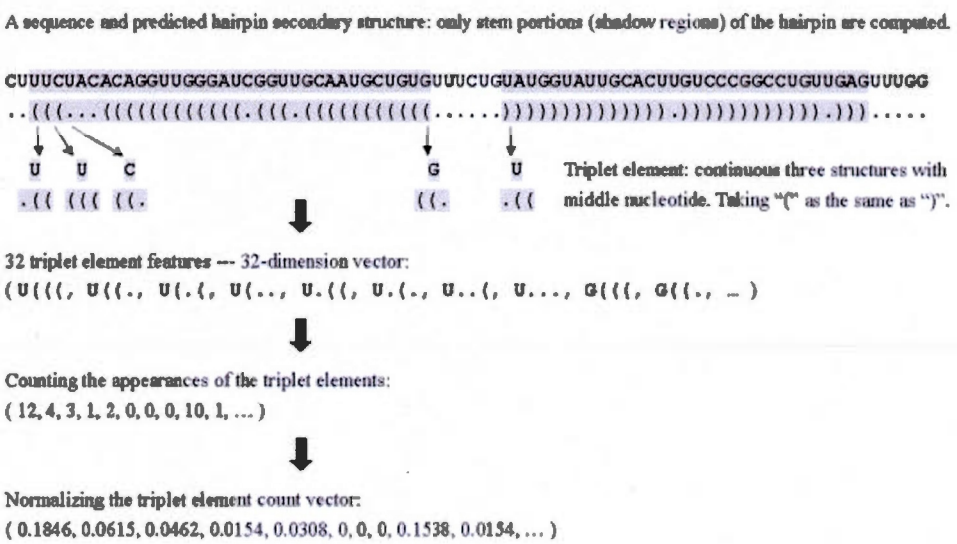


Figure 2.7 Étapes des créations des éléments du vecteur dans *Triplet-SVM*

(Xue et al. 2005)

MiPred commence par déterminer si la MFE obtenue peut l'être seulement par conformation aléatoire. Pour cela, une simulation Monte-carlo basée sur un test de permutation détermine une p-value. Pour calculer une p-value, le test se déroule comme suit :

1. Calculer la MFE originale de la structure du précurseur
2. Changer l'ordre des nucléotides aléatoirement dans la séquence originale et recalculer la MFE
3. Refaire l'étape précédente un grand nombre de fois (1000) de manière à construire une distribution stationnaire des valeurs de MFE possibles
4. Si N est le nombre d'itérations et R est le nombre de séquences qui ont subi des changements aléatoires dont la MFE est inférieur ou égal à la MFE originale, alors la p-value P est définie par :

$$P = \frac{R}{N+1} \qquad (4)$$

Tableau 2.5 Classement de l'importance relative des caractéristiques par *MiPred*

(Jiang et al. 2007)

Rang	Caractéristique	Précision
1	*P*-value	15.80
2	MFE	5.48
3	C...	2.04
4	U(((	2.00
5	A(((	1.49
6	A...	0.83
7	G...	0.76
8	U..(	0.43
9	G.((	0.34
10	A(..	0.31
11	C((.	0.31
12	G..(	0.29
13	G(..	0.29
14	U((.	0.27
15	U...	0.26
16	U(.(	0.24
17	G(((	0.23
18	C(((	0.20
19	A..(	0.20
20	U(..	0.19
21	C(..	0.14
22	U.(.	0.14
23	G((.	0.09
24	C.(.	0.09
25	A(.(	0.08
26	C..(	0.08
27	C.((	0.07
28	A.(.	0.07
29	G(.(	0.06
30	A.((	0.03
31	G.(.	0.02
32	A((.	0.00
33	U.((	0.00
34	C(.(	0.00

Ensuite, *MiPred* emploie des forêts d'arbres décisionnels (Breiman 2001), qui sont des algorithmes d'apprentissage d'aide à la décision et à l'exploration de données. Enfin, pour le problème de la prédiction, *MiPred* effectue une classification. À cet effet, il calcule la précision de la prédiction totale (ACC pour Accuracy), la spécificité (Sp), la sensitivité (Se), et le coefficient de corrélation de Mathew (MCC) (Matthews 1975). Ce calcul s'effectue comme suit :

$$ACC = \frac{VP + VN}{VP + VN + FP + FN} \times 100\%$$
$$Sp = \frac{VN}{VN + FP} \times 100\%$$
$$Se = \frac{VP}{VP + FN} \times 100\% \quad (5)$$
$$MCC = \frac{VP \times VN - VP \times FN}{\sqrt{(VP + FP) \times VN + FN) \times (VP + FN) \times (VN + FP)}} \times 100\%$$

Où FP correspond aux faux positifs, VP sont les vrais positifs, FN sont les faux négatifs et VN sont les vrais négatifs.

L'exécution de miPred avec l'exemple 1557_1102_758 donne le fichier au contenu suivant:

```
AUGAAAAACAAGGGAUUAGCAAGUUUGAGCUCGGUGGUAGAAUUGCUCAAUGAGAUCGUGAAUAGGAG
CAUGAUUCAACCAAUUGACAUCAAUGUUGAUAAA  102    Yes
..............(((((((...((((..(((((((((.(((((((((.(((....))).....)))
))..))))).))).)))))..))))))))))...  -23.30            0.094 pseudo
      73.4%
```

Ce résultat présente le précurseur sélectionné par mipred sur la séquence b présentée en 2.4.1 (Voir Figure 2.6). La séquence a été coupée afin d'en extraire la meilleure épingle à cheveu, sa MFE a été recalculée et deux scores apparaissent : La p-value, 0.094, et le résultat de classification, pseudo 73.4%. C'est donc une pseudo-épingle à cheveu prédite à 73.4% avec une e-value inférieure à 0.1. Dans le cas inverse, *pseudo* est remplacé par *real*. La structure secondaire est schématisée dans la Figure 2.8. Si on la compare avec la Figure 2.6b, il est facile de distinguer visuellement quelle partie a été extraite par miPred.

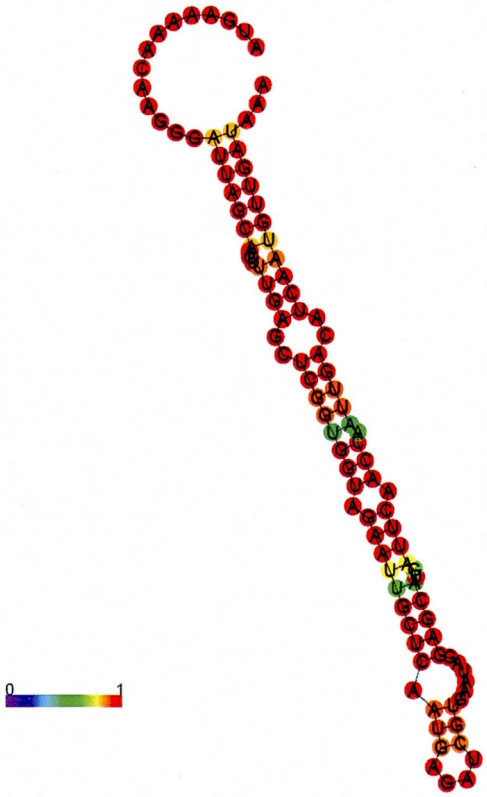


Figure 2.8 Exemple de structure de précurseur sélectionnée par miPred

2.4.2.2 *HHMMiR*

La méthode *HHMMiR* (Kadri et al. 2009) exploite la puissance des modèles de Markov cachés hiérarchiques. Elle construit un modèle de Markov caché à partir de la structure des microARNs comme montré en Figure 2.9.

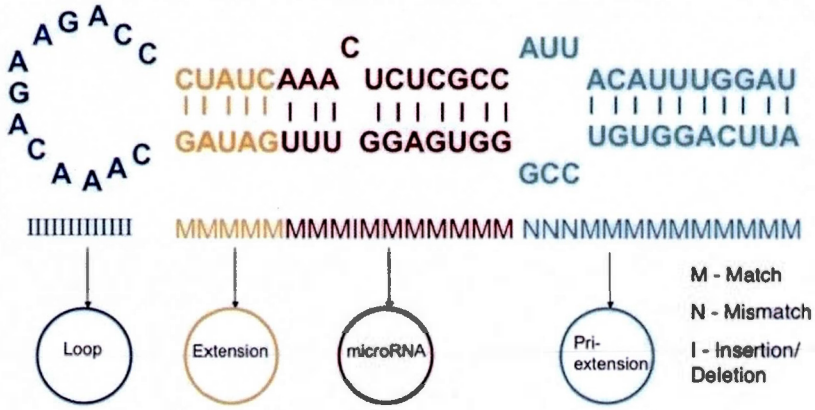


Figure 2.9 L'épingle du microARN selon *HHMMiR*

(Kadri et al. 2009)

Le modèle obtenu est hiérarchique et présente trois niveaux essentiels (voir Figure 2.10). L'état épingle à cheveux, *Hairpin*, est le nœud de départ, et ne peut avoir de transition qu'avec l'état boucle, *loop*. Dans le modèle, toute épingle commence avec une boucle. Les quatre états internes au second niveau correspondent aux quatre régions principales de l'épingle à cheveux. Ce niveau possède aussi un état fin, *end*, permettant de revenir à l'état épingle à cheveux. Chaque état interne a un modèle de probabilité au niveau inférieur. Une boucle ne peut pas avoir de bases pairées et donc n'a qu'un seul sous état : I, pour insertion-deletion. L'état "Extension" ne peut émettre qu'un état de correspondance de bases après avoir quitté la boucle puisqu'un mésappariement fera partie de cette dernière. Enfin, chaque état possède une *fin*. Dans la Figure 2.10, les formes ovales représentent les états internes du modèle, les couleurs correspondent aux régions représentées dans la Figure 2.9, les petits cercles en ligne continue sont les états possibles dits de production et ceux en pointillés symbolisent les états de fin, dits silencieux. M: état de correspondance, *Match*, N: état de mésappariement, *Mismatch*, I: insertion-délétion, L_{end}: état de fin de la boucle, R_{end}: état de fin du microARN, P_{end}: état de fin du pri-extension.

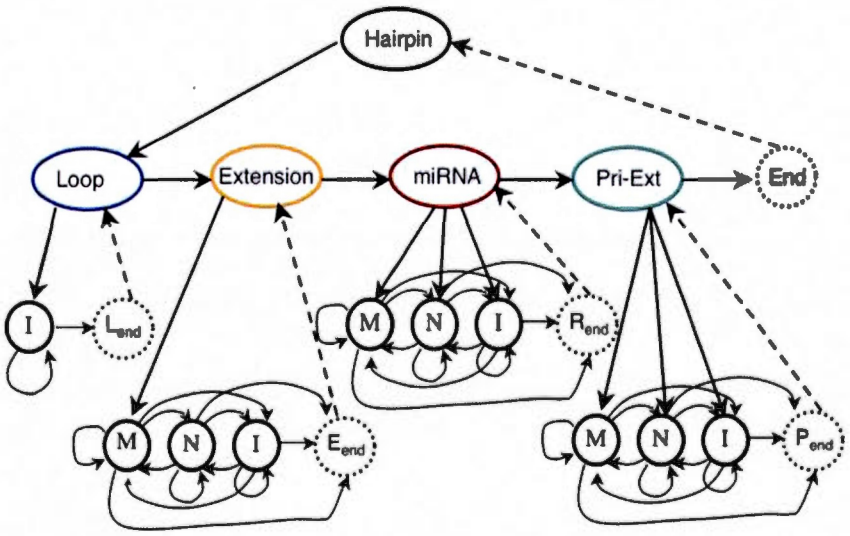


Figure 2.10 Etats du modèle HHMM (basé sur un microARN)

(Kadri et al. 2009)

Le résultat de l'exécution de HHMMiR pour la séquence couleur d'exemple 1557_1102_758 donne un fichier dont le contenu est présenté ici:

```
>taggagcatgattcaaccaa_3721,AUUAGCAAGUUUGAGCUCGGUGGUAGAAUUGCUCAAUGAGA
UCGUGAAUAGGAGCAUGAUUCAACCAAUUGACAUCAAUGUUGAUAAA,(((((((...((((..((((
((((.(((((((((.(((....))).....))))))..)))).))).))))..)))))))))))...
,-98.06363801791193,-104.42897680958214,0.939046239979188,TRUE
```

Tout comme miPred, HHMMiR coupe aussi l'épingle à cheveux qu'il considère comme la meilleure. La séquence obtenue n'est pas exactement la même que miPred, on peut le constater en comparant la Figure 2.8 à la Figure 2.11. Cette dernière est issue de la séquence *b* de la Figure 2.6. Deux chiffres sont donnés, -98 et -104 arrondis, ils correspondent respectivement au log de la vraisemblance calculée en fonction des modèles déterminés à partir de faux précurseurs (négatif) et celui de vrais précurseurs (positif). Un ratio des deux est donné, 0.94, qui permet de déterminer si le précurseur peut être considéré comme vrai ou faux, en fonction du respect du modèle. Plus ce chiffre est haut, moins le modèle positif est respecté. Une limite de ce ratio doit être appliquée, elle peut être obtenue à partir de plusieurs tests sur des données connues, telles que les précurseurs publiés sur miRbase.

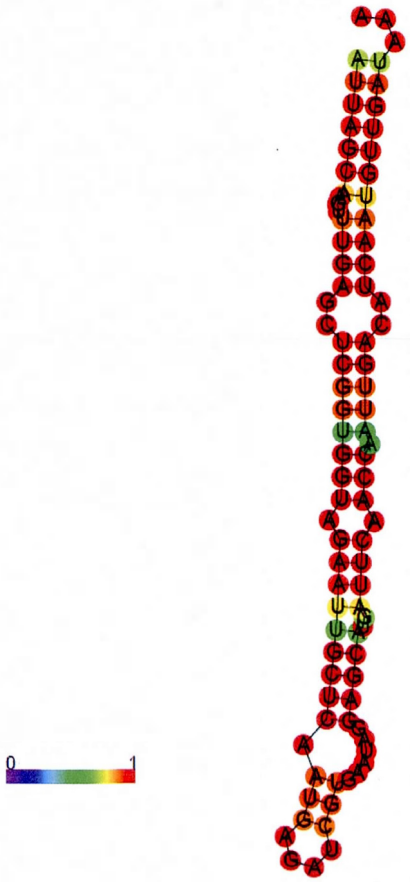


Figure 2.11 Exemple de structure de précurseur sélectionnée par HHMMiR

2.5 Prédiction des gènes cibles par *TAPIR*

La fonction d'un microARN est de pouvoir cibler un ARN messager pour en stopper la traduction en protéine. Ainsi, la recherche des gènes cibles a une double fonction, celle de valider la prédiction d'un microARN et celle de découvrir ses cibles potentielles. Les cibles du microARN caractérisent sa fonction. Un microARN prédit qui ne possède pas de cible est considéré comme étant un faux positif ou un microARN dont la cible n'est pas présente dans les gènes analysés. La recherche des gènes cibles s'effectue donc juste après l'étape de prédiction des précurseurs de microARNs.

Les algorithmes de prédiction de gènes cibles se basent sur des propriétés telles que la thermodynamique de l'appariement, la complémentarité miRNA:mRNA, ou encore la conservation en 3'. Dans tous les cas, les logiciels prennent en entrée des séquences de microARN, et des séquences de référence correspondant aux ARN messagers. Une faiblesse de ces algorithmes est le haut taux de faux positifs par rapport à la réalité. Ceci s'explique par une compréhension limitée des processus biologiques associés au mécanisme de liaison entre un microARN et son gène cible. Quelques bases de données qui stockent des informations relatives aux gènes cibles des microARNs déjà connus existent, telles que miRDB (Wang 2008), miRecords (Xiao et al. 2009) et Tarbase (Sethupathy et al. 2006).

Il existe plusieurs programmes pour déterminer les gènes cibles d'un microARN, spécifiques à la recherche de gènes cibles chez les plantes ou chez les animaux. Pour les plantes, il existe *TAPIR* (Bonnet et al. 2010), *miRU* (Zhang 2005) et *psRNATarget* (Dai & Zhao 2011). Pour les animaux, la diversité de logiciels est plus importante, où nous retrouvons *miRanda* (John et al. 2004), *TargetScan* (Lewis et al. 2003), *PicTar* (Krek et al. 2005) *miTargets* (Kim et al. 2006) et *DIANA microT* (Maragkakis et al. 2009).

A l'heure actuelle, deux choix s'offrent aux utilisateurs pour calculer les gènes cibles de leurs microARNs. Ils peuvent soit effectuer leurs calculs sur un serveur distant, en ligne, ou localement si le programme est fourni. Dans notre cas, vu l'ampleur des calculs à effectuer, nous ne pouvions pas utiliser de version en ligne. Donc, dans nos travaux, nous avons opté pour *TAPIR*, qui fait partie des logiciels les plus récents, rapide en vitesse de calcul et dont le code source est téléchargeable.

TAPIR utilise un outil appelé "*alignement local fasta*" qui permet une recherche rapide contre les ARN messagers. Toutefois il ne prend pas en charge les appariements (duplexes) comprenant des hernies ou des mésappariements. Un autre outil, plus lent, est fourni dans *TAPIR* : *RNAhybrid* (Rehmsmeier et al. 2004). C'est un algorithme plus efficace et plus précis car il tient compte des hernies et des mésappariements. Il se base sur la programmation dynamique, où il calcule l'énergie libre minimum des hybridations de toutes les positions de départ possibles du microARN sur sa séquence cible. Des restrictions sur la longueur ou les hernies peuvent être fournies. *RNAhybrid* tient aussi compte de la recherche du mimicry (Franco-Zorrilla et al. 2007), caractérisé par la présence d'une grosse hernie d'environ trois

nucléotides localisée sur le site de clivage habituel retrouvé chez les plantes. Ce mimicry est généralement situé au nucléotide 10 ou 11 du gène cible.

Un score est établi par *TAPIR* dépendamment du nombre de nucléotides du duplexe qui occasionneraient une hernie, des mésappariements ou des paires de G-U (appariement possible à la place de A-U habituel).

Le score *S* déterminé par *TAPIR* prend en compte le nombre de mésappariements, le nombre de gaps (introduits par des hernies et des boucles) et le nombre de paires G-U. En plus, plusieurs études ont montré l'importance des germes (*seed*) de la région du duplexe dans la partie 5' du microARN. Cette région est significativement appauvrie en mésappariements et hernies pour les cibles valides des microARNs. Ainsi une pénalité supplémentaire sur les mésappariements, gaps et paires de G-U entre les paires 2 à 12 de la séquence du microARN permettent d'inclure ce constat :

$$S = Nm + Ng + (0.5 \times Nu) + (2 \times Nsm) + 2 \times Nsg) + Nsu \qquad (6)$$

où Nm, Ng et Nu correspondent respectivement au nombre de mésappariements, gaps et paires de G-U en dehors de la région du *seed*. Nsm, Nsg et Nsu sont les mêmes paramètres mais dans le *seed*.

Ci-bas le résultat fourni par *TAPIR* pour la séquence couleur d'exemple 1557_1102_758. Plusieurs caractéristiques sont dévoilées dans cette illustration, tels que la cible, le score, la position de départ sur la cible, les paires de G-U, etc. Dans cet exemple nous observons une complémentarité de score 0, donc parfaite, sur l'EST CJ871514 à la position 441 :

```
miRNA          UAGGAGCAUGAUUCAACCAAU
target         CJ871514
score          0
mfe_ratio      1.00
start          441
seed_gap       0
seed_mismatch  0
seed_gu        0
gap            0
mismatch       0
gu             0
miRNA_3'       UAACCAACUUAGUACGAGGAU
aln            |||||||||||||||||||||
target_5'      AUUGGUUGAAUCAUGCUCCUA
```

CHAPITRE III

APPROCHES BIOINFORMATIQUES POUR PRÉDIRE DE NOUVEAUX MICROARNS

Dans ce chapitre, nous présentons brièvement les approches actuelles à partir de données de séquençage haut débit. Puis, nous exposons notre approche de prédiction de microARNs exploitant les principaux outils décrits au chapitre II. Une vue d'ensemble des résultats obtenus et une analyse sommaire de leur expression différentielle sont aussi présentés.

3.1 Approches actuelles pour prédire de nouveaux microARNs à partir de séquençage à haut débit

Depuis que l'existence des microARNs est bien établie, de nombreuses études ont permis d'en identifier plusieurs milliers répartis dans diverses espèces (Hammell 2010, Volinia et al. 2010), que ce soit chez les animaux (Jin et al. 2009) ou les plantes (Jasinski et al. 2010, Jin et al. 2008, Sunkar et al. 2008, Szittya et al. 2008, Wei et al. 2009). À cause des nombreuses méthodes existantes pour toutes les étapes intermédiaires qui mènent à la prédiction de nouveaux microARNs, il n'existe aucune approche standard actuellement. Nous abordons brièvement cette question, en proposant la plateforme Armadillo (en annexe A). Ici, nous nous intéressons à la variété des méthodes utilisées. En effet plusieurs études partent d'un séquençage de petits ARN et tentent de découvrir de nouveaux microARNs en exploitant ou non la conservation de ces petits ARN entre plusieurs espèces (Lindow et al. 2007). Dans ces

études, seul un logiciel de prédiction de précurseur est utilisé, et les gènes cibles ne sont pas toujours identifiés. Une autre approche employée dans plusieurs autres études consiste à se focaliser sur les microARNs conservés entre les espèces en alignant tous les microARNs publiés sur une espèce particulière (Yao et al. 2007). Dans ce cas-ci, seuls de nouveaux candidats des familles connues sont retrouvés.

Au début de ce chapitre, nous présentons deux articles associés aux récentes techniques d'identification de microARNs à partir des séquençages à haut débit. Ces deux articles sont celui de Schulte et collaborateurs (Schulte et al. 2010) et de Lindow et collaborateurs (Lindow et al. 2007). Dans ces deux exemples, les étapes de recherche des microARNs ont été réalisées de manière différente. Chacune s'adapte aux données de départ (séquençage ou banque d'ARNm) et aux objectifs de l'étude. Dans la première étude, un séquençage à haut débit sur la plateforme AB SOLID a révélé une expression différentielle de microARNs dans un neuroblastome (tumeur touchant les cellules embryonnaires). Les auteurs ont séquencé le transcriptome de cinq patients ayant un neuroblastome dit favorable (non actif) et cinq autres non favorable (actif). Ils ont effectué cette analyse pour rechercher les microARNs qui pourraient agir en tant que gènes oncogènes, ou au contraire, suppresseurs de tumeurs. Pour en arriver à cette fin, ils ont d'abord retiré les adaptateurs des séquences couleurs par le programme *cutadapt* qui a été créé par les auteurs. Les fragments inférieurs à 35pb ont été alignés par *MAQ* sur le génome humain et comptés pour extraire les niveaux d'expression selon les échantillons de départ. Les quantifications ont été normalisées par transformation linéaire. À partir d'un échantillon de référence choisi arbitrairement, une distribution de points quantile-quantile (qq) des fragments ayant une expression d'au moins 5 au total dans toutes les librairies a été établie. En parallèle, une partie de l'ARN séquencé a été transcrit à l'inverse par *transcriptase inverse* afin d'analyser l'édition de l'ARN (changements posttranscriptionnels de l'ADN). Afin de découvrir les microARNs inconnus potentiellement présents dans leur séquençage, une version du programme miRdeep (Friedlander et al. 2008) a été modifiée et utilisée pour extraire les microARNs en fonction de l'abondance et la distribution des fragments séquencés sur les précurseurs.

Dans un autre contexte, l'équipe de Lindow a exploité la correspondance intragénomique de microARNs potentiels dans plusieurs espèces pour prédire de nouveaux microARNs et leurs cibles. La correspondance intragénomique consiste à trouver les mêmes microARNs et leurs

cibles correspondantes entre plusieurs espèces. Pour cela ils ont développé leurs propres outils intégrés dans un pipeline appelé *miMatcher*. Ce pipeline comprend *miSVM*, *miHomology* et *miSquare*. *miSVM* est un SVM entrainé sur quelques caractéristiques structurales des précurseurs de miRbase, et accepte en entrée des précurseurs. *miHomology* permet de vérifier si deux précurseurs qui proviennent d'espèces différentes sont homologues. Pour cela, il se base principalement sur la conservation du microARN porté dans d'autres espèces et la longueur de la tête d'épingle. Enfin, *miSquare* détermine les protéines orthologues entre des espèces ainsi que les cibles des microARNs conservés.

Trois espèces de plantes ont servi de base à cette étude : *Arabidopsis thaliana*, *Populus trichocarpa*, et *Oryza sativa*. Ils ont remarqué que les correspondances intragénomiques entre ces espèces avec une approche d'apprentissage supervisé contenaient suffisamment d'informations pour permettre une prédiction fiable de microARNs, sans qu'il y ait conservation entre eux. En utilisant cette méthode, ils ont identifié 1200, 2500 et 2100 microARNs potentiels dans *A. thaliana*, *P. trichocarpa*, et *O. sativa* respectivement. Le pipeline *miMatcher* fonctionne de la manière suivante :

- La recherche des micro-correspondances initiales avec *vmatch* (Kurtz 2003), en acceptant 2 mésappariements sur une taille de correspondance minimale de 20 nucléotides ;
- Le filtrage des correspondances intragénomiques en fonction des conditions suivantes :
    - L'énergie libre de liaison par base, calculée par *RNAcofold* (Vienna) (doit être d'au moins -1,4kcal/mol) ;
    - La structure secondaire du précurseur (extraction de 10 bases avant le microARN potentiel et 240 après, puis calcul de la structure par *RNAfold*). La structure doit contenir une épingle à cheveux avec le microARN potentiel dans un des bras.
- Récupérer l'épingle cible dans le long précurseur.
- Enfin, de nombreux paramètres sont pris en compte pour vérifier si le précurseur est un vrai précurseur capable de porter un microARN. Ces paramètres sont par exemple le nombre de paires de bases dans l'épingle, l'énergie de repliement, la présence de boucles et de hernies, etc.

Par la suite, les séquences génomiques disponibles et des transcrits d'ARNm pour chaque espèce sont placés en entrée et soumis à une correspondance intragénomique. Les résultats de cette correspondance génèrent des micro-correspondances composées de paires de segments de génome et d'ARNm, qui sont par la suite comparées avec les microARNs de chaque espèce publiés sur miRbase, puis soumises à *miSVM* et *miHomology*. Des filtres sont ensuite appliqués pour détecter les microARNs qui ont au moins un homologue dans un des deux autres organismes. Enfin, les protéines orthologues entre des espèces sont déterminées par *miSquare* et le nombre de microARNs candidats conservés est présenté.

Toutefois, ces différentes approches ne répondent pas aux défis qui nous sont imposés dans notre étude. En effet, une complexité de la problématique actuelle que nous étudions dans ce projet est liée au fait que nous travaillons à partir des ESTs du blé, composés uniquement de séquences codantes. Il est donc difficile d'exploiter directement les méthodes construites pour les génomes complets, ni celles basées sur la conservation. Ensuite, rares sont les études sur la découverte de microARN qui emploient la plateforme SOLID pour obtenir leurs données de séquençage, et ce notamment chez les plantes. La plupart des méthodes de prédictions existantes proviennent de l'apprentissage machine et nécessite des données réelles d'entraînements ou sont déjà paramétrées pour des espèces ou des familles définies. Ainsi, il est rare de trouver un logiciel qui s'applique aux plantes en général et au blé en particulier. De plus, après plusieurs tests, nous avons déterminé que la prédiction des précurseurs de microARN par différents programmes propose dans la majorité des cas des épingles différentes. Il parait donc crucial de ne pas se fier au résultat d'un seul logiciel de prédiction comme effectué par les papiers décrits plus haut. Enfin, il n'existe aucun logiciel capable de valider la position du microARN sur le précurseur. Ce qui nous parait pourtant comme un critère essentiel. L'ensemble de ces défis sont relevés dans les sections qui suivent.

3.2 Nouvelle approche bioinformatique pour prédire de nouveaux microARNs chez le blé à partir de ses ESTs

Préambule : Cette partie présente la version actuelle de notre article en préparation pour publication. Des validations expérimentales des microARNs prédits sont en cours. L'étude a

été menée en collaboration avec Mesdames Agharbaouiz Zahra et Fahiminiya Somayyeh, Monsieur Amine Remita, Pr. Houde Mario et sous la supervision de Pr. Diallo Abdoulaye Banirẽ, et Pr. Sarhan Fathey. Madame Agharbaoui Zahra a cultivé les plants et procède à la validation expérimentale des microARNs. L'analyse de l'expression différentielle des microARNs a été produite par Madame Fahiminiya Somayyeh et Monsieur Amine Remita. Pr. Houde Mario a contribué à apporter son expertise dans la définition des protocoles expérimentaux.. Pr. Diallo Abdoulaye Baniré et Pr. Sarhan Fathey ont conçu et dirigé l'étude. Enfin, j'ai produit le pipeline bioinformatique, programmé toutes ses étapes et produit les principaux résultats de prédiction.

3.2.1 Introduction

Dans cette étude, nous recherchons les microARNs du blé impliqués dans différents processus biologiques permettant à cette espèce de se défendre contre quelques stress, tels que l'exposition au froid, à l'aluminium et au sel. D'un point de vue biologique, à l'heure actuelle, de nombreux facteurs de transcription qui permettent au blé de tolérer ces stress ont été identifiés et caractérisés. Cependant de nouveaux mécanismes de régulation ont été identifiés récemment et impliquent les microARNs. Mais le rôle et le mode d'action de ces derniers restent à définir (Wei et al. 2009). Un autre défi majeur concerne l'annotation fonctionnelle des milliers de gènes du blé (Romeuf et al. 2010), où une étude approfondie des ARN ciblés par les microARNs peut permettre de mettre à jour de nouvelles fonctions si ces gènes sont déjà annotés. Cependant, ces recherches ne peuvent pas s'effectuer sans avoir recourt à la bioinformatique. En effet, un séquençage global de tous les fragments d'ARNs ayant une taille similaire aux microARNs génère une énorme quantité de données qui ne peut être gérée manuellement. De plus, il faudrait plusieurs millions de dollars pour déterminer si toutes les petites séquences d'ARN sont des microARNs. Ainsi, l'utilisation des approches bioinformatiques combinatoires ou d'apprentissage machine qui permettent de prédire si un fragment séquencé est un microARN est d'une grande importance. Cependant, comme présenté dans les chapitres précédents, la mise en place de ces approches n'est pas toujours facile. Car chaque étape de prédiction du processus traite de problèmes informatiques qui sont difficiles à résoudre.

Ainsi, pour répondre à la problématique d'identification et de caractérisation des microARNs impliqués dans des stress abiotiques donnés, il faudrait identifier tous les microARNs du blé issus du séquençage ainsi que leurs gènes cibles, les regrouper en famille en fonction de leur similarité en séquence et leurs cibles communes. Puis, il faudrait définir dans quelle(s) condition(s) de stress abiotiques ils sont impliqués. Pour ceux dont l'information fonctionnelle a pu être identifiée, il est important de produire une relation entre les différents stress et les fonctions biologiques à travers l'exploitation des ontologies de gènes.

Dans cete étude, trois variétés de blé ont été cultivées sous des conditions environnementales induisant un stress pour la plante. Les stress induits sont l'exposition à de l'aluminium, du sel, du gel et/ou du froid. Les variétés choisies sont connues pour avoir une certaine adaptabilité à ces expositions (Delhaize et al. 2009, Guy 1990, Saqib et al. 2008, Sarhan et al. 2006). À partir de ces plants, les ARNs ont été extraits avec le protocole d'Ambion (Austin, Texas). Ensuite, grâce à un séquençage sur la plateforme ABI SOLID, un total 89 millions de fragments de 35pb de longueur repartis en dix librairies (conditions) ont été obtenus. Parmi ces fragments, nous prédisons et annotons ceux qui sont des microARNs à partir de notre nouvelle approche bioinformatique. Dans les sections qui suivent, nous présentons cette approche qui a permis la prédiction de 1016 nouveaux microARNs potentiels chez le blé. La différence avec les méthodes traditionnelles existantes est qu'ici nous exploitons deux logiciels de prédiction et d'extraction des précurseurs (HHMMiR et MiPred) et ne gardons que les communs (à cause du haut taux de faux positifs de ces méthodes). Généralement, ces logiciels de prédictions employés dans les approches traditionnelles, analysent seulement la structure de l'épingle à cheveux pour effectuer une prédiction sans tenir compte de l'évidence du microARN séquencé. Indiquer qu'un fragment séquencé est un microARN seulement en fonction du précurseur n'est qu'une condition minimale dans la biogenèse des microARNs. Donc, il faudrait effectuer des validations subséquentes. Une première étape consiste à se concentrer seulement sur le fragment séquencé à l'intérieur du précurseur. Puis, analyser l'ensemble des fragments en compétition sur le même précurseur en fonction de leur abondance et identifier des cibles pour ces microARNs. Ainsi, nous avons développé un outil de prédiction en post-traitement de l'approche traditionnelle, que nous avons nommé *miRdup*, "dup" comme duplexe. Il se concentre exclusivement sur la position du fragment séquencé sur son précurseur pour

déterminer si cette position est celle d'un valide microARN. À cette fin, nous avons entrainé plusieurs classifieurs en apprentissage machine. Nous avons fourni une dizaine de paramètres qui caractérisent le duplexe du microARN et sa région complémentaire sur l'épingle à cheveu du précurseur. Cette région complémentaire contient également le microARN*. Le microARN* se retrouve généralement parmi les fragments séquencés. Après une sélection des meilleures caractéristiques, les classifieurs ont été évalués sur tous les microARNs publiés dans les bases de données publiques (miRbase, PMRD) dans le but de déterminer le meilleur modèle à implanter ici. Une fois ce choix effectuer, nous avons prédit les gènes cibles des microARNs potentiels en utilisant le programme TAPIR (Bonnet et al. 2010). Puis, nous avons analysé l'expression différentielle des microARNs en fonction des différents stress analysés à partir d'une analyse de Z-score avec une correction pour le taux de fausses découvertes (False Discovery Rate) à 0.05 en utilisant la méthode de Benjamini-Hochberg (Benjamini & Hochberg 1995). Enfin nous regroupé les principales caractéristiques fonctionnelles des microARNs à partir de leurs ontologies et nous avons vérifié la présence de ces microARNs dans d'autres espèces. Les sections qui suivent décrivent le matériel utilisé, les données générées, les méthodes implantées, les résultats obtenus et une brève discussion de ces résultats qui sont encours de validations expérimentales.

3.2.2 Matériel et méthodes

3.2.2.1 *Culture des plantes, purification des petits ARNs et préparation des librairies*

Nous avons construit dix librairies distinctes de trois variétés de blé hexaploïde de printemps et d'hiver en phase végétative ou reproductive et exposés à différentes conditions de stress qui sont le froid, l'exposition au sel et à l'aluminium. Le Tableau 3.1 montre la composition des librairies (*L*) sur des variants de blés Clair (*CL*), Bounty (*BT*) et Atlas (*AT*), provenant soit de racines (*r*) ou de feuilles (*l*) et sous des conditions normales, ou exposés au froid (*v*), sel (*s*) et aluminium (*al*). Le variant Clair est un blé tolérant au froid et à la salinité, tandis

que Bounty est sensible au froid, salinité et aluminium. Atlas est un variant tolérant à l'aluminium. Dans chaque cas, 200mg de tissu ont été prélevés.

Tableau 3.1 Détails de la composition des librairies et des conditions

Variété	Librairie et abréviation	Conditions/Traitement
Clair (Tolérant)	*L1CLnc*	Tissus aériens à partir de plants en phase végétative sous des conditions normales
	L2CLv	Tissus aériens à partir de plants en phase végétatives vernalisées à 4°C de 2 à 24 heures, pendant 1 à 7 semaines
	L3CLr	Tissus aériens à partir de plants en phase reproductive (Plants vernalisés désacclimatés sur 3 à 5 semaines)
	L4CLsl	Tissus aériens de 4 vieux plants exposés à 200mM de NaCl
	L5CLsr	Tissus racinaires de 4 vieux plants exposés à 200mM de NaCl
Bounty (sensible)	*L6BTnc*	Tissus aériens à partir de plants en phase végétative sous des conditions normales
	L7BTc	Tissus aériens à partir de plants en phase végétative acclimatés au froid pendant 4 semaines
	L8BTal	Tissus racinaires à partir de plants exposés à 5µM d'aluminium
Atlas (Tolérant)	*L9ATnc*	Tissus racinaires à partir de plants sous des conditions normales
	L10ATal	Tissus racinaires à partir de plants exposés à 50µM d'aluminium

***nc*= conditions normales. *v*= vernalisation. *r*=racines. *sl*=salinité dans les feuilles. *sr*=salinité dans les racines. *c*=froid. *al*=aluminium.**

Tous les petits ARNs exprimés (en dessous de 200pb) ont été isolés des plants soumis aux conditions exposées dans le Tableau 3.1 à partir du kit d'isolation *Ambion mirVana*. La concentration et quantification des petits ARNs extraits ont été déterminées par *spectrophotomètre Nanodrop 1000* et *Bioanalyzer*, et leur purification a été effectuée avec *flashPAGE fractionation* d'*Ambion*. Les librairies ont été construites en utilisant le protocole *SREK* (Small RNA Expression Kit, Ambion), et les ADNc obtenus ont été sélectionnés selon leur taille (100-150pb) et purifiés. Après un dernier contrôle des tailles et qualité des librairies, les séquences ont été quantifiées et normalisées par *qPCR*, réparties en concentration équimolaires et enfin séquencées par la plateforme *SOLID Analyzer* (V2.1 d'*Applied Biosystem*) tel que décrit dans la section 2.1.2

3.2.2.2 *Nouveau pipeline de la prédiction des microARNs et de leurs gènes cibles à partir des fragments séquencés*

La Figure 3.1 présente une vue d'ensemble du pipeline développé pour cette étude. Les sections qui suivent décrivent en détail les étapes et les comptages des données obtenues après chaque étape de prédiction et de filtrage. Sur cette figure, les cases en vert représentent les comptages uniques, en jaune les comptages complets contenant des duplications, en bleu le compte des précurseurs uniques, en rouge les microARNs uniques, et en gris le compte des microARNs uniques supprimés. En résumé, le pipeline commence par le regroupement de 1.4 millions d'ESTs provenant de plusieurs bases de données, ainsi que des 89 millions de séquences couleurs séquencées par SOLID. Les adaptateurs sont supprimés par *cutadapt* et ceux non compris entre 16 et 31pb sont supprimés. Un alignement des fragments est réalisé sur les ESTs avec *MAQ*. Les petits ARNs exprimés moins de 5 fois dans toutes les librairies sont supprimés. Les précurseurs et les séquences des petits ARNs sont ensuite extraits des ESTs en se basant sur la position de l'alignement. La composition des fragments en ARNs conservés et non codants est analysée avec *BLAST* contre les bases de données de microARNs et d'ARNs non codants. Les précurseurs sont ensuite repliés avec *RNAfold* et prédits par *HHMMiR* et *miPred* comme vrai ou faux précurseurs de microARNs. Les gènes cibles sont recherchés avec *TAPIR* contre les ESTs, et un *BLAST* est de nouveau effectué contre les bases de données de microARNs. À la fin, plusieurs analyses permettent de filtrer les prédictions, en supprimant ceux qui n'ont pas de gènes cibles, ou qui ont une ressemblance avec des ARNs ribosomaux ou non codants, ou qui ne remplissent pas les conditions de *mirDup*. De l'information additionnelle est ajoutée, telle que la classification des microARNs prédits en familles, les microARNs*, l'analyse de l'expression différentielle entre les librairies, les ontologies des gènes ciblés, et l'organisation en catégories.

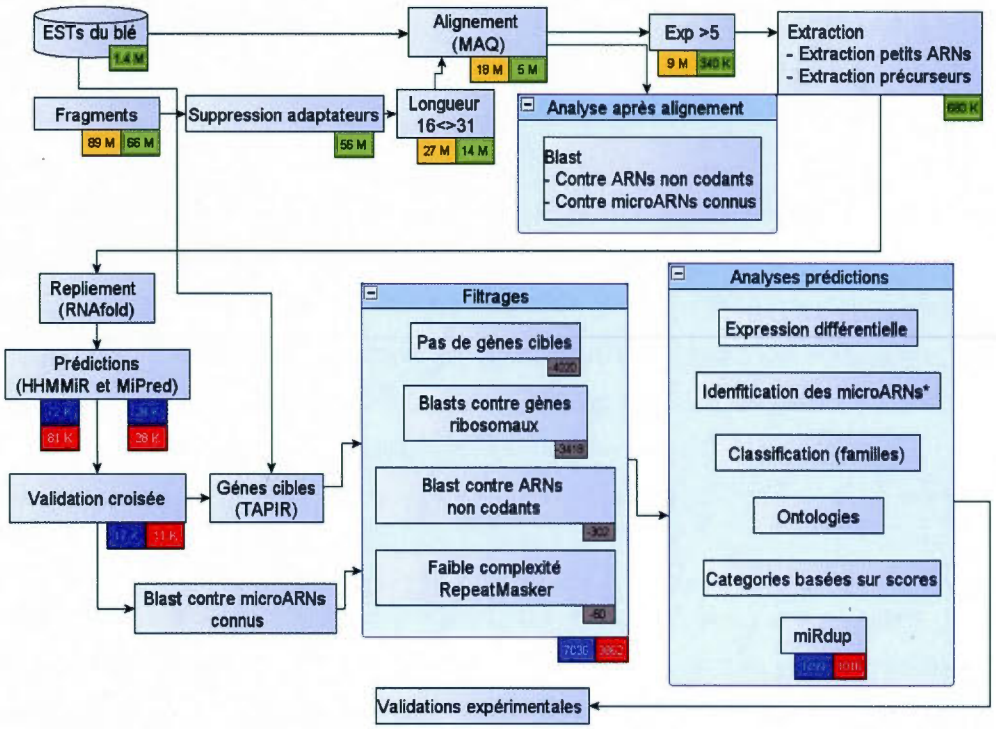


Figure 3.1 Pipeline des étapes de la prédiction des microARNs et leur analyse

3.2.2.2.1 Approche de la suppression des adaptateurs et alignement

Lorsque des fragments sont obtenus d'un séquençage *AB SOLID*, ils sont associés à une mesure de confiance de prédiction appelée *qualité des fragments*. Ainsi, avant tout, la qualité du séquençage a été vérifiée directement en parcourant les fichiers de qualité produits par *SOLID*. Les valeurs de qualités (QV) de fragments séquencés sont comprises dans un intervalle de 0, dit faible, à 33, meilleure (voir section 2.1.2 pour plus de détails). Généralement, le séquenceur a une meilleure performance d'exactitude de lecture sur les premiers nucléotides. Ainsi plusieurs études (Hackenberg et al. 2009, Ribeiro-dos Santos et al. 2010) considèrent qu'un fragment est de bonne qualité si sa QV est au moins supérieure à 10 pour toutes les dix premières bases (QV_{10}<10). Pour notre étude, nous n'avons pas souhaité supprimer dès le départ les fragments qui n'étaient pas de bonne qualité.

Ainsi, nous avons pu tester la robustesse de notre pipeline en analysant le pourcentage de fragments de bonne qualité au début et à la fin du pipeline.

Avant l'alignement, pour obtenir la séquence des petits ARNs séquencés, les adaptateurs ont été supprimés des séquences couleurs avec le programme *cutadapt* v0.9 (Schulte et al. 2010) et adaptés à *MAQ* v0.7.1 (Ondov et al. 2008) pour l'alignement. Comme conseillé dans l'étude de Schulte (Schulte et al. 2010), un maximum de 12% de mésappariements sur l'adaptateur a été considéré. Après l'alignement, les précurseurs ont été extraits simplement en récupérant une séquence plus longue autour de l'endroit où le petit ARN avait été aligné sur les ESTs. Nous avons envisagé le cas où le petit ARN pouvait se trouver sur le bras droit du précurseur en récupérant -160pb avant la position de départ de l'alignement sur le génome, et 20pb après la position de fin de l'alignement. Sur le bras gauche, c'est l'inverse. Ces bornes de coupures sont plus grandes que la taille des précurseurs connus. Cependant, une fois repliés, si ces séquences contiennent des structures en épingle à cheveux, alors ces structures sont extraites. Puis, il est vérifié que les fragments séquencés sont contenus dans ces épingles.

Des statistiques sur la distribution des petits ARNs après la suppression des adaptateurs et l'alignement ont été calculées pour chaque librairie. L'ensemble des données réparties dans les librairies contient de nombreuses répétitions, reflétant les profils d'expression de toutes les séquences selon les conditions de stress de la plante. Étant donné que la littérature indique une taille de microARNs variant entre 16 et 31pb, nous avons retenu seulement les séquences qui avaient ces tailles après la suppression de l'adaptateur. Après l'alignement, toutes les séquences qui ont au moins un mésappariement, pour les petits ARNs de 16 à 18pb de long, ou deux mésappariements, pour ceux entre 19 à 31pb, ont été identifiées. Parmi ceux identifiés, nous avons supprimé les séquences qui ont un faible niveau d'expression (inférieur à 5) dans les 10 librairies.

3.2.2.2.2 Méthode de calcul de la composition des fragments

La composition des fragments doit être analysée afin de détecter tout d'abord si les séquences obtenues sont connues dans les bases de données de microARNs et d'ARNs non codants. Dans un premier temps, après le retrait des adaptateurs, afin d'évaluer le potentiel en microARNs des fragments, nous avons analysé le taux de présence des fragments séquencés

dans les bases de données de miRbase et PMRD. Le programme d'alignement local *BLAST* (Altschul et al. 1990) a permis d'identifier les microARNs conservés connus et présents dans nos fragments. Tous les appels de *BLAST* ont été exécutés avec les paramètres suivants : *word_size* 4, *maximum e-value* 0.1, *percentage identity* 80, *gap open penalty* 5, *gap extension penalty* 2, *match score* 1, *mismatch score* -3, *filter low complexity*. Pour empêcher les alignements aléatoires, des restrictions ont été ajoutées. Pour être retenu, chaque résultat doit avoir une couverture du petit ARN séquencé de 100% pour les séquences ayant une longueur inférieure ou égale à 16 (et minimum 90% pour les autres longueurs), une couverture du sujet (microARN de la base de données) de minimum 90%, et un nombre de gaps extérieurs inférieur à trois. Tous les résultats obtenus sont présentés dans un arbre produit par l'outil en ligne *iTol* (Letunic & Bork 2007) afin de déterminer le degré de conservation des microARNs entre les différentes espèces contenues dans les deux bases de données.

Par la suite, nous avons aussi utilisé *BLAST* sur les petits ARNs contre la base de données non codante *Noncode v2.0* (Liu et al. 2005) afin d'identifier les petits ARNs non codants connus présents dans nos fragments.

Enfin, *RepeatMasker v3.2.9* (Smit et al. 1996) a été utilisé avec *RepBase15.09* (Jurka 1998) et *Repeatmasker-Libraries20100611* afin d'éliminer les petits ARNs de faible complexité, incluant les séquences répétées.

3.2.2.2.3 Prédiction des microARNs et leurs cibles à partir des précurseurs potentiels

Les précurseurs potentiels prélevés ont tous été repliés avec *RNAfold*, du paquet *Vienna v1.8.4* (Hofacker et al. 1994) avec les paramètres par défaut.

Pour prédire si le repliement d'un précurseur peut potentiellement porter un microARN, les logiciels *HHMMiR* (Kadri et al. 2009) avec la plateforme *Java* v.1.6 et *miPred* (Jiang et al. 2007) avec le paquet *R-2.1.01.1* ont été choisis. Il était important de choisir deux logiciels qui emploient des algorithmes d'apprentissages différents, afin de diversifier les prédictions et éviter des biais quelconques que pourrait apporter l'une ou l'autre méthode. Les microARNs prédits par les deux techniques auront donc satisfait deux modèles bien distincts.

Pour *HHMMiR*, les jeux de données d'entrainement ont été construits à partir de *miRbase* et *PMRD*. Si l'épingle sélectionnée par les deux programmes ne contient pas le fragment séquencé en entier, alors ce précurseur n'est pas considéré dans la suite des analyses.

Pour les microARNs potentiels prédits par *HHMMiR* et *miPred*, nous avons identifié leurs gènes cibles à partir du programme *TAPIR* v1.0 (Bonnet et al. 2010). Ce dernier se paramètre par la valeur d'un score calculé et de la MFE. Le manuel du programme conseille un score maximum de 3 et une MFE supérieure ou égale à 0.7. En effet, les résultats qui ont un score très élevé, supérieur à 3, n'ont pas été retenus car ils pourraient représenter des duplexes qui possèdent trop de mésappariements. Les microARNs potentiels qui n'ont pas obtenu de gènes cibles ont été retirés car ils ne respectaient pas le principe de fonctionnement connu de ceux-ci. Cependant, il faut retenir que nos données sont partielles. Ainsi, plusieurs microARNs pourraient ne pas avoir leurs cibles parmi les ESTs utilisés.

En fonction des gènes cibles obtenus pour chaque microARN potentiel, l'annotation fonctionnelle de ce dernier a été réalisée à partir de la banque de données des ontologies des gènes (*GO : Gene Ontology*). Les ESTs ciblés ayant une référence *uniProt* (Bairoch et al. 2005) dans leur annotation ont été récupérées avec *GOretriver/GOSlim* (Fontana et al. 2006) sur *AgBase* (McCarthy et al. 2006). Cette dernière étape de l'analyse a permis de classifier les gènes cibles en fonction de la hiérarchie des fonctions biologiques des gènes ciblés par les microARNs.

3.2.2.2.4 Nouvelle approche d'apprentissage machine de validation de la position du microARN sur un précurseur prédit

Afin de créer un modèle de validation basé sur de l'apprentissage machine, nous avons utilisé *Weka* et ses librairies (Hall et al. 2009). Ce qui facilite ainsi la sélection des attributs et la création des modèles. Pour le classifieur SVM, qui est supervisé, nous avons utilisé la librairie *libSVM* (Chang & Lin 2011). Afin d'entrainer les modèles de prédiction, deux jeux de données furent nécessaires. Un jeu de données positif qui contient des duplexes valides et un autre qui contient des duplexes qui ne sont pas de vrais microARNs (appelé jeu de données négatif). Les deux jeux ont été extraits de *miRbase 14*, qui contient 14197 microARNs pour 22824 précurseurs. Tous les précurseurs ayant plusieurs boucles n'ont pas été pris en compte puisque nos modèles de prédiction ne les supportent pas. 18615

précurseurs avec leur microARN respectif. Le test négatif a été généré avec les mêmes données, excepté que la position du microARN a été changée aléatoirement sur le précurseur. Leu jeu de données pour le contrôle exploite les microARNs et les précurseurs provenant de la base de données PMRD, qui contient 8680 précurseurs.

Le modèle est construit sur la base d'attributs détaillés dans le Tableau 3.5. Ils sont déterminés en fonction des éléments qui caractérisent le duplexe. Cependant l'influence de ces attributs varie en fonction de l'importance qu'on leur accorde. En effet, plusieurs d'entre eux présentent des données similaires entre les jeux de données d'entrainement positifs et négatifs. Afin d'éliminer les attributs de faible importance, nous avons utilisé un évaluateur d'attribut, *Info Gain* (Menzies et al. 2007), qui évalue la valeur d'un attribut en mesurant l'information obtenue par rapport à la classe d'attributs. La méthode de recherche est un ordonnanceur, *ranker*, qui ordonne les attributs selon leur évaluation individuelle en conjonction avec les évaluateurs d'attributs. La valeur du rang représente le « mérite moyen » de cet attribut. Parmi les résultats nous avons choisi d'ignorer les caractéristiques qui ont un rang inférieur à 0.05. Car les rangs faibles pourraient biaiser la classification.

Les temps de calcul des différentes évaluations ont été mesurés sur un ordinateur Intel Core 2 de 2.66 GHz quatre cœurs et 6GB RAM. Les MFE des duplexes miRNA-miRNA* ont été obtenus avec *RNAduplex*, du paquet *Vienna* (Hofacker et al. 1994).

La sensitivité et la spécificité des classifieurs ont été calculées avec les formules classiques :

$$sensitivité = \frac{vrais\ positifs}{vrais\ positifs + faux\ négatifs}$$

$$specificité = \frac{vrais\ négatifs}{vrais\ négatifs + faux\ positifs}$$

3.2.2.2.5 Méthodes d'analyse statistique pour l'identification des microARNs différentiellement exprimés

L'abondance des microARNs reflète leur activité dans une cellule (Yang et al. 2005). En comparant les librairies entre elles nous pouvons avoir des renseignements sur les niveaux d'expression des microARNs. Afin de quantifier et de comparer l'expression de chaque

microARN dans l'ensemble des librairies, le nombre de fragments a été normalisés en *rpm* (reads per million) en fonction du nombre total de séquences dans chaque librairie. Les séquences qui ont une abondance totale inférieure à 5 ont été ignorées. Les fragments qui ont des séquences couleurs conduisant à la même séquence nucléotidique ont été combinés par addition de leurs abondances distinctes.

Pour déterminer les microARNs qui ont une expression différente entre les librairies créées à partir des plantes témoins (qui ont poussé en conditions normales), et celles exposées aux stress abiotiques ou à partir de plantes à deux stades de développement différents, la procédure suivante dérivée des analyses de micropuces a été appliquée. L'expression différentielle entre les microARNs exprimés dans plusieurs librairies a été normalisée en fonction de leurs distributions respectives. Pour cela, la procédure standard de normalisation a été effectuée. Puis un score Z (Z-score) (Kal et al. 1999) a été calculé pour chaque expression différentielle. Pour minimiser l'impact des faux positifs, nous avons appliqué la correction *FDR* (False Discovery Rate (Benjamini & Hochberg 1995)). Ainsi, nous obtenons une p-value associée à chaque expression différentielle. Seuls les expressions différentielles avec une p-value corrigée inférieure à 0.05 ont été retenues. Étant donné un microARN quelconque; n_1 et n_2 - les abondances de ce microARN dans deux librairies L_1 et L_2, et N_1 et N_2 - le nombre total de microARNs dans les deux librairies respectives. Le score Z (Z-score) de la différence d'expression est calculé selon la formule suivante pour diminuer le biais de la disparité d'abondance entre librairie (Kal et al. 1999) :

$$Z = \frac{p_1 - p_2}{\sqrt{p_0(1-p_0)(\frac{1}{N_1}+\frac{1}{N_2})}} \qquad (7)$$

où $p_1 = \frac{n_1}{N_1}, p_2 = \frac{n_2}{N_2}$ et $p_0 = \frac{n_1+n_2}{N_1+N_2}$.

3.2.2.2.6 Méthode de regroupement des microARNs en famille

Les microARNs prédits ont été regroupés en familles par rapport à leur séquence mature. Pour cela, leur alignement multiple a été calculé avec *ClustalW2* (Larkin et al. 2007) et une classification hiérarchique a été générée. Les longueurs de branches de l'arbre issu de la classification ont permis de déterminer si les clades regroupant plusieurs microARNs représentaient une famille. Pour obtenir le seuil de décision de groupage, une classification

hiérarchique des microARNs de miRbase a été effectuée. Puis, le seuil de séparation des familles de miRbase en fonction des longueurs de branches de l'arbre obtenu a été estimé pour obtenir environ le même nombre théorique de familles dans miRbase. Ce seuil, de 0.15285, a servi comme facteur discriminant pour nos microARNs aussi. L'avantage de cette approche de regroupement est relié au fait qu'elle tient compte des gènes cibles. Car les séquences très similaires ciblent les mêmes gènes.

3.2.2.3 *Ensemble d'outils et de programmes développés pour la gestion des données*

Cette étude a généré un total de plus de 100 gigas octets de données diverses au cours des différentes étapes du pipeline de recherche. Afin de pouvoir les gérer et mener à bien toutes les étapes, de l'alignement jusqu'aux analyses des microARNs prédits, les programmes écrits ont fait appel aux objets java sérialisés. Dans l'annexe, nous avons placé les principaux scripts. Avant d'utiliser les objets, nous avons organisé tous les résultats en fichiers textes simples, mais les accès incessants aux fichiers pour mettre à jour les données à chaque étape rendaient la tâche trop lente. Nous avons aussi opté pour un système en base de données, qui malgré son efficacité était difficilement portable et tout aussi lent que les fichiers. Avec une organisation des données dans des objets, beaucoup plus simple, ces problèmes se sont résolus. Aussi, les programmes ont été codés de manière à faciliter leur intégration dans le programme de gestion de flux nommé *Armadillo* présenté en annexe.

3.2.2.3.1 Programmes écrits

Tous les scripts décrits doivent être exécutés dans l'ordre sur un nouveau jeu de données de séquençage :

1. a_cutadapt

Les programmes contenus dans cette section traitent les étapes après séquençage, dont la suppression des adaptateurs (*cutadapt*) et l'alignement.

a. ESTsCleaning.java

Ce programme a pour but de nettoyer l'ensemble des ESTs regroupés de plusieurs bases de données. De nombreux doublons sont présents une fois le regroupement des sources et il a été nécessaire de les enlever pour éviter les faux duplicatas lors de l'alignement et la recherche des gènes cible.

b. i_*cutadapt*ToFasta.java

Converti les fichiers FASTQ après *cutadapt* vers un format FASTA. Le format FASTQ est particulier car il contient les informations de qualité des fragments, et pour une lecture plus facile des séquences il doit être converti.

c. ii_extractSmallrnasByLength.java

Le logiciel d'alignement *MAQ* a la particularité de traiter seulement des fichiers qui contiennent des séquences de taille unique. À partir des fichiers FASTQ créés par *cutadapt,* plusieurs fichiers sont créés selon la taille des séquences. Au final, il y aura dix dossiers, pour dix librairies, et chaque librairie contiendra seize fichiers (contenant les séquences de longueurs 15 à 30pb). Cette dispersion des données a l'avantage de pouvoir être parallélisés aisément sur des plateformes de calcul.

d. iii_getExpression.java

Ce programme permet d'obtenir le niveau d'expression de chaque séquence couleur unique dans toutes les librairies. Ce niveau d'expression renseignera sur l'abondance d'un petit ARN selon le stress induit à la plante

e. iv_adaptExpression.java

Ce programme sert à convertir le format des expressions, en changeant les virgules en tabulations et en calculant le total dans toutes les librairies pour des analyses en parallèle.

f. v_parseDumpFile.java

Le programme *MAQ* met dans un fichier *dump* l'ensemble des résultats multiples de l'alignement. Les meilleurs résultats sont dans un autre fichier (mapview). Ce programme permet simplement d'extraire les informations pertinentes, telles que la localisation de l'alignement, la source et les mésappariements.

g. vi_getSequencesPrecursorsFromDump.java

Une fois que le fichier dump a été réorganisé, ce programme récupère les séquences dans les ESTs ainsi que leurs précurseurs.

h. vii_getSequencesPrecursorsFromMapview.java

Le fichier *mapview* provenant de *MAQ* contient les informations sur les meilleurs résultats de l'alignement des fragments contre les ESTs (génome). Ce programme récupère les séquences du séquençage ainsi que les précurseurs correspondants.

i. ix_adaptFolding.java

Quelques ESTs contenaient encore des N dans leur séquence, et comme les logiciels de prédiction ne pouvaient les prendre en compte, les séquences ont été coupées au niveau des N avant de les replier. Ce programme permet de vérifier dans quelle partie de séquence coupée et repliée se trouve le petit ARN séquencé. D'autre part, il recréé la structure du fichier qui n'était plus en format FASTA après le repliement.

2. b_alignement

Les programmes de cette section apportent des analyses en parallèle des étapes de recherche à partir des données alignées.

a. analyseMapping.java

La majorité des scripts sont exécutés sur à l'aide d'un *LSF* (Load Sharing Facility qui est un répartiteur de tâches sur un cluster de calcul), et les sorties d'écran de chacun d'entre eux sont stockées dans des fichiers. La sortie d'écran de *MAQ* contient des informations sur les statistiques de l'alignement, récupérées par ce programme.

b. mappingQuality.java

Afin d'obtenir une idée de la qualité du séquençage, ce programme contrôle la qualité de chaque fragment qui a été aligné. Il calcule combien de bases ont une qualité inférieure ou égale à 10 dans les 10 premières bases.

c. getUnmappedReads.java

Ce programme récupère les fragments qui n'ont pas alignés contre les ESTs du blé. Sachant que les ESTs ne contiennent que les régions codantes de l'ADN, il manque potentiellement ceux des introns et des régions intergéniques. Un alignement de ces fragments contre d'autres espèces proches du blé et ayant un génome complet permettrait d'obtenir d'autres microARNs.

3. d_genescibles

Cette section comprend les programmes qui traitent les données obtenues après le calcul des gènes cibles.

a. i_parse_*TAPIR*.java

Ce programme lit les fichiers de sortie de *TAPIR* afin d'en extraire les résultats sous un format plus facilement exploitable et rajoute les annotations des gènes ciblés.

b. ii_geneOntology.java

Certaines annotations des ESTs contiennent des identifications de référence à des bases de données protéiques (uniprot). Afin de pouvoir identifier quel processus biologique est ciblé par le microARN, Ce programme extrait les références utiles pour ensuite chercher leurs fonctions biologiques à partir de l'ontologie de gènes GO.

c. iii_miRNAsTargetGenesStats.java

Ce programme recherche les statistiques des gènes cibles qui ont des références protéiques ou non, qui sont inconnus, et identifiables par d'autres noms.

4. e_filtres

Les programmes de cette section filtrent les microARNs ribosomaux et de faible complexité.

a. i_parseRibosomaux.java

Ce programme permet d'extraire les informations nécessaires à la reconnaissance de gènes ribosomaux. Des Blasts ont été effectués sur NCBI contre toutes leurs bases de données

génomiques pour chercher si nos microARNs ressemblaient à des ribosomaux. Les sorties de NCBI sont en format *xml*, une extraction des blasts positifs est donc exercée par ce programme.

b. ii_parseLowComplexityRM.java

Afin de faciliter l'organisation des données, les fichiers en FASTA sous forme de >nom\nSequence ont souvent été écrits sous la forme >Sequence\nSequence. Or, certains logiciels comme Repeat Masker refusent des séquences à la place de nom. Les séquences ont donc été numérotées, traitées par Repeat Masker et ce programme a refait la correspondance des noms et numéros.

5. f_analyses

Cette section regroupe les programmes qui exécutent différentes analyses sur les données acquises à la fin des étapes du pipeline.

a. check*BLAST*s.java

Ce programme parcourt les fichiers de *BLAST* et accepte certains matchs comme vrais ou faux selon nos paramètres présentés dans la section matériel et méthodes de ce mémoire.

b. checkRNAComplements.java

Ce programme vérifie la présence du complément ou du complément inverse dans un jeu de séquences. Ceci a été réalisé dans les données alignées et celles après prédiction dans les fichiers FASTA de sortie de *cutadapt*.

c. conservedAgainstMirnasDBsItol.java

Ce programme génère les fichiers acceptables par *Itol* pour construire l'arbre de conservation des microARNs sur différentes espèces.

d. i_mirnasStars.java

Ce programme permet de prédire un microARN* à partir du microARN, du précurseur et de la structure secondaire.

e. ii_checkMirnasStarsIn*Cutadapt*.java

Ce programme vérifie la présence des microARNs* calculés précédemment dans le séquençage.

f. iii_conservedAgainstNCrnas.java

Ce programme aurait pu être plus haut dans la hiérarchie puisqu'il sert à extraire les statistiques de *BLAST*s des petits ARNs alignés mais non prédits contre les non codants. Les non codants possèdent plusieurs catégories, et il faut déterminer combien d'ARN nous avons dans ces catégories.

6. g_familles

Dans cette section, les programmes classent en famille les microARNs prédits.

a. checkNumberFamiliesMiRbase.java

Ce programme compte le nombre de familles présentes dans *miRbase*. Ce repère du nombre de familles est important pour le programme suivant.

b. getFamiliesFromClustalTree.java

Les microARNs prédits doivent être classés en familles. Pour cela, un alignement multiple est construit pour toutes les séquences avec le logiciel ClustalW. Le cladogramme fourni est ensuite interprété par ce programme afin de grouper les microARNs et obtenir une classification. L'algorithme se base sur la mesure de distance entre les branches pour déterminer les clades. La mesure de distance limite entre " même clade" ou "nouveau clade" est mesurée sur un test positif sur miRbase dans le programme précédent

7. outils

Les programmes de cette section servent exclusivement à organiser en objets les données acquises au long des étapes du pipeline.

a. i_add*MAQ*ResultsToObject.java

Ce programme regroupe les résultats de sortie de *MAQ* pour les stocker dans des objets. Il est exécuté après tous ceux de a_*cutadapt*.

b. ii_insertPredictionInObjects.java

Programme à exécuter après les prédictions. Il met à jour les objets précédemment créés pour y ajouter les prédictions.

c. iii_addTargetsAndAnalysesToObjects.java

Ce programme est à exécuter après les gènes cibles. Il met à jour tous les objets après prédiction. Il regroupe de nombreuses autres analyses telles que : ribosomaux, conservés, groupes, non codants, faible complexité, microARN*, familles, qualité des fragments, MFE des précurseurs recalculés, les expressions concaténées, et les ontologies.

Deux codes restants ne sont pas exécutables : rnaobject2.java, qui décrit l'objet serialisable, et tools.java, qui contient plusieurs fonctions souvent appelées par les différents scripts.

3.2.2.3.2 Informations générées pour chaque microARN prédit

Pour chaque microARN prédit, plusieurs informations ont été acquises au cours du processus, comme montré dans le Tableau 3.2. Ces informations sont très variées. Elles sont extraites au fur et à mesure de l'avancée dans le pipeline. Tous les programmes utilisés livrent de nombreux résultats en sortie, dont les plus utiles et pertinents pour notre étude sont conservés à des fins d'analyses ou pour la suite du pipeline.

Tableau 3.2 Informations produites pour chaque microARN au cours de cette étude

Information	Détails	Type de donnée
Abondance du microARN*	Abondance du microARN* dans le séquençage	`Entier`
Annotation de l'EST	Annotation de l'EST source de l'alignement	`String`
Bras	Bras sur lequel le microARN est localisé sur le précurseur	`String`
Brin	Brin positif ou négatif de l'EST source (+ ou -)	`Chararactère`
Catégorie	Catégorie attribuée pour classer les	`Double`

	microARNs selon les scores de prédiction	
Commun	Commun à la prédiction de *HHMMiR* et *MiPred*	`Booléen`
Conservé	microARN prédit présent dans les bases de données miRbase et PMRD	`Booléen`
Couleur	Séquence couleur du fragment, issue du séquençage	`String`
Différentiellement exprimé	Si le microARN est différentiellement exprimé entre des librairies, si oui lesquelles	`String`
Est un microARN	Est oui ou non un microARN	`Booléen`
Faible complexité	Le microARN prédit est détecté par Repeat Masker en tant que séquence de faible complexité	`Booléen`
Famille	Famille (Issu de la classification calculée sur l'arbre de distance)	`String`
Liste des gènes cibles	Liste des gènes cibles	`Table de strings`
Logiciel de prédiction	*HHMMiR* ou *MiPred*	`String`
Longueur du petit ARN	Longueur du petit ARN après alignement	`Entier`
Longueur du précurseur	Longueur du précurseur	`Entier`
Logiciel d'alignement	Seulement *MAQ*	`String`
Mésappariement ajouté	Mésappariement ajouté lors de la récupération du nucléotide manquant	`Booléen`
Mésappariements	Mésappariements lors de l'alignement	`Entier`
MFE du précurseur	Energie libre du précurseur	`Double`
microARN réél ou pseudo selon *MiPred*	microARN réel ou pseudo microARN	`String`
microARN*	microARN* calculé par rapport à la structure secondaire	`String`
miRdup	Prédit comme vrai par miRdup, qui valide la position du microARN sur le précurseur	`Booléen`
Niveaux d'expression corrigés	Étant donné que des fragments différents peuvent aboutir sur plusieurs séquences	`Table d'entiers`

	nucléotidiques, les niveaux d'expression de ces séquences sont additionnés	
Niveaux d'expression originaux	Niveaux d'expression originaux, compte total dans les dix librairies	`Table d'entiers`
Nom	Nom du fragment, issu du séquençage	`String`
Nom du ribosome	Nom du ribosome si ribosomal	`String`
Nombre de gènes cibles	Nombre de gènes cibles	`Entier`
Nombre de précurseurs uniques	Un microARN peut provenir de plusieurs précurseurs	`Entier`
Non codant	Le microARN prédit a blasté contre la base de données nonCode	`Booléen`
Notes	Commentaires	`String`
Numéro d'accession de l'EST	Numéro d'accession de l'EST source de l'alignement	`String`
Ontologie, composant cellulaire	Composant cellulaire de Gene Ontologie à partir des uniRefs des gènes cibles annotés	`String`
Ontologie, fonction moléculaire	Fonction moléculaire de Gene Ontologie à partir des uniRefs des gènes cibles annotés	`String`
Ontologie, ID de la protéine	ID de la protéine de Gene Ontologie à partir des uniRefs des gènes cibles annotés	`String`
Ontologie, Nom de la protéine	Nom de la protéine de Gene Ontologie à partir des uniRefs des gènes cibles annotés	`String`
Ontologie, processus biologique	Processus biologique de Gene Ontologie à partir des uniRefs des gènes cibles annotés	`String`
Position du petit ARN	Position du petit ARN séquencé et aligné sur l'EST source	`Entier`
Qualité	Qualité du fragment, correspondant aux nombre de bases séquencées ayant une qualité inférieure à dix dans les dix premières bases	`Entier`
Ribosomal	Le microARN prédit est aussi classé sur ncbi dans les ribosomaux	`Booléen`
Score de confidence de *MiPred*	Score de confidence de *MiPred* en pourcentage	`Double`
Score de prédiction	Score de prédiction (P-Value pour *MiPred*,	`Double`

	ratio de vraisemblance pour *HHMMiR*)	
Séquence couleur sans adaptateur	Séquence couleur après que l'adaptateur eut été supprimé	`String`
Séquence du petit ARN après alignement (microARN)	Séquence complète du microARN	`String`
Séquence du précurseur	Séquence nucléotidique du précurseur	`String`
Structure du précurseur	Structure du précurseur parenthésée	`String`
Total des niveaux d'expression	Total des niveaux d'expression	`Entier`
Total des niveaux d'expression corrigé	Total des niveaux d'expression corrigé	`Entier`
Uniref	Uniref (uniprot)	`String`

String = Chaine de charactères

3.2.3 Résultats et discussion

Pour déterminer si les fragments d'ARN séquencés par des séquenceurs de nouvelle génération sont de vrais microARNs, une approche informatique plus précise que celles utilisées actuellement dans la littérature a été mise en place. Elle consiste à combiner les résultats de deux programmes de prédiction dans une approche de validation croisée et à ajouter une étape de validation en post-traitement des logiciels de prédiction des pré-microARNs. L'alignement a été réalisé en deux étapes, peu employé dans la littérature, avec une suppression des adaptateurs, évitant ainsi les faux mésappariements.

L'identification de microARNs chez les plantes, tout comme chez les animaux, à partir des précurseurs peut être effectuée de multiples façons en fonction des technologies choisies pour agencer les différentes étapes du pipeline. Nous pouvons utiliser différents séquenceurs (e.g. SOLID, Illumina ou Roche 454), programmes d'alignement (e.g. *MAQ*, *Bowtie*, *BFAST*, *BLASTX*), logiciels de repliement (e.g. *RNAFold*, *McFold*, *MFold*), ou logiciels de prédiction de précurseurs (e.g. *HHMMiR*, *miPred*, *Triplet-SVM*) et gènes cibles (e.g. *TAPIR*, *psRNATarget*). Toutefois les méthodes existantes dans la littérature à chaque niveau du pipeline ont leurs particularités et donnent souvent des résultats différents entre elles. Il serait

intéressant de faire une revue de l'efficacité de toutes ces méthodes en effectuant une comparaison, mais cela va au delà de l'objectif de cette maitrise.

À chaque étape, nous avons suivi l'évolution des fragments issus du séquençage en comptant le nombre de redondants et d'uniques. Ils sont décrits dans les sections qui suivent et sont affichés dans le pipeline de recherche au point 3.2.2.2 D'autre part, afin de mieux visualiser les données obtenues, un exemple du fragment 1557_1102_758 est donné dans les différentes parties du chapitre II ainsi que dans le Tableau 3.13.

3.2.3.1 *Séquençage, distribution des tailles et alignement*

Le séquençage a généré 89 105 096 fragments en séquence couleur initiaux, parmi lesquels 66 400 401 fragments uniques. L'adaptateur a été retrouvé dans 56 437 315 fragments. Parmi ces derniers, seulement 27 152 240 séquences, 13 939 779 uniques avaient une taille comprise entre 16 et 31pb.

Une analyse de la qualité du séquençage a été vérifiée sur l'ensemble des fragments sur toutes les librairies. Les résultats sont exposés dans le Tableau 3.3. Après le séquençage, plus de 80% des fragments avaient au maximum 2 bases ayant une valeur de qualité inférieure à 10. Après les prédictions, à la fin du pipeline, 85% des fragments n'ont aucune base de mauvaise qualité sur les 10 premières bases. Les étapes de l'étude ont donc éliminé naturellement une grande fraction des fragments de mauvaise qualité.

Tableau 3.3 Mesure de qualités des fragments séquencés

$QV_{10}<10$	Moyenne sur toutes les librairies après séquençage (%)	Moyenne sur toutes les librairies après prédictions (%)
0	56,15	84,29
1	15,17	10,92
2	10,42	3,49
3	6,52	0,76
4	4,70	0,36
5	2,72	0,11
6	1,84	0,03
7	1,03	0,03
8	0,97	0,01

9	0,31	0,00
10	0,17	0,00

La Figure 3.2 montre la répartition des séquences par taille après la suppression des adaptateurs. Dans la partie A de cette figure, la distribution des longueurs est en accord avec la connaissance actuelle dans la littérature (Schulte et al. 2010), avec deux pics aux positions 21 et 24pb. Le pic à 35pb représente la portion des séquences dont l'adaptateur n'a pu être retrouvé, et les fragments de taille 1 sont ceux où l'adaptateur a été retrouvé en début de séquence. En marron, nous avons identifié ceux dont la taille est comprise entre 16 et 31pb. En jaune, sont présentés d'autres types d'ARN ou des produits de dégradation. Les parties B et C reflètent les disparités dans les distributions entre les librairies, de manière visuelle dans un *heatmap* et plus précise sous forme de courbes. Les numéros des librairies et les acronymes des parties B et C sont expliqués dans le Tableau 3.1.

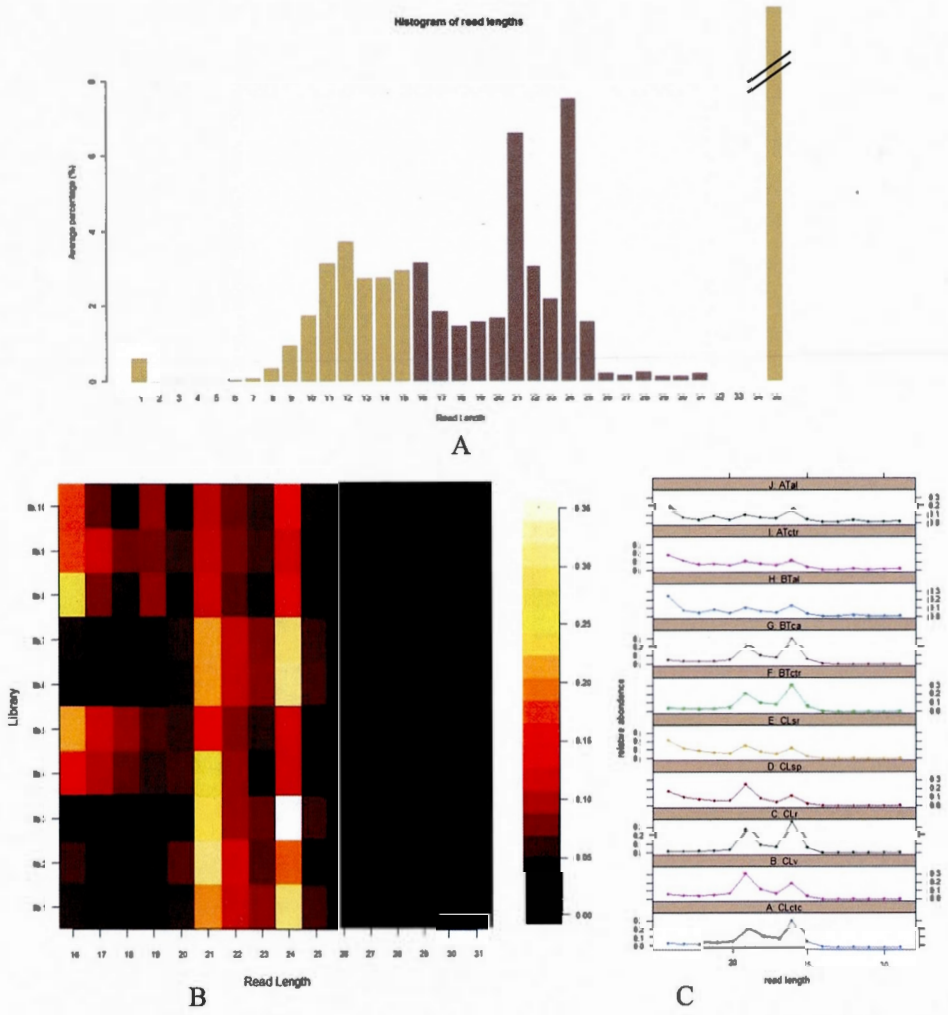


Figure 3.2 Distributions des séquences après suppression de l'adaptateur

A : histogramme de toutes les librairies concaténées. B : Heatmap des distributions de taille pour chaque librairie. C : Graphiques de distributions de taille pour chaque librairie

Toutes ces distributions montrent des différences significatives entre les librairies qui peuvent être expliquées par la profondeur du séquençage, la taille du génome, le stade de développement ou le type de tissu. La fréquence de taille des fragments après la coupure de l'adaptateur varie de 0 à 34pb en couleur (1 à 35 pour la séquence nucléotidique). Ces résultats sont similaires à ceux obtenus dans une étude sur l'*Arabidopsis Thaliana* (Song et al. 2010) ou même chez l'humain (Schulte et al. 2010).

L'alignement est l'une des étapes cruciales du processus de prédiction, et ce pour plusieurs raisons. Premièrement, la séquence couleur livrée par le séquenceur ABI SOLID ne permet pas d'avoir une séquence nucléotidique directement. Or une seule erreur de séquençage dans une base quelconque, rendrait inexact tout le reste de la séquence. Par ailleurs, ce format est complexe à manipuler lorsque l'on cherche à valider à la main les étapes avec des petits échantillons de données. Il faut noter que cette plateforme a été très peu utilisée pour analyser des transcriptomes (Cloonan et al. 2008, Tang et al. 2009). Deuxièmement, l'utilisation d'un génome partiel composé seulement de séquences codantes constitue une autre source importante de limitation de l'alignement. Étant donné que le génome de blé est actuellement séquencé mais non assemblé, il n'y a que des ESTs qui sont disponibles. Ces derniers ont dû être récupérés sur plusieurs bases de données de laboratoires répartis dans le monde qui travaillent sur le séquençage du blé. Troisièmement, les résultats de plusieurs programmes d'alignement ont donné des résultats très différents. Durant cette étude, quatre d'entre eux ont initialement été testés, *RNA2MAP*, *SHRiMP*, *Bioscope* et *MAQ*. Concernant les paramètres de *RNA2MAP*, des essais ont été effectués en alignant les fragments d'une librairie contre les génomes complets d'*Arabidopsis Thaliana* et d'*Oryza Sativa*. Les paramètres qui ont donné les meilleurs résultats sur ces jeux de données connus ont été choisis pour aligner les fragments contre les ESTs du blé. Cependant le nombre de fragments alignés était trop faible. Pour le programme *SHRiMP*, nous avons fait face à une limitation inattendue puisqu'il ne peut aligner des séquences dont la taille est inférieure à 24pb pour les raisons exposées en 2.3.2 *MAQ* en combinaison avec *cutadapt*, donne les taux de d'alignement plus élevés.

Afin de tester la fiabilité de tous ces programmes et faire notre choix pour l'étude, nous avons donc calculé le nombre de fragments en séquences couleurs qui sont alignés par chacun d'entre eux, ainsi que ceux alignés en commun. Ces résultats sont présentés dans le diagramme de *Venn* de la Figure 3.3. Dans cette figure, les intersections des résultats montrent des abondances très faibles entre les programmes d'alignement, ils n'alignent donc pas les mêmes fragments. De plus, sachant que plus de 89 millions de fragments ont été séquencés, Bioscope et SHRIMP n'ont aligné que 2.3% et 4% respectivement. RNA2MAP montre de meilleurs scores, mais la très grande majorité des fragments alignés étaient inférieurs à 17 nucléotides et intégraient jusqu'à 6 mésappariements. En descendant la limite

à 3 mésappariements, seuls 19 millions de fragments ont pu être alignés, soit environ 21%. Cependant, la majorité est située en dessous de 17 nucléotides.

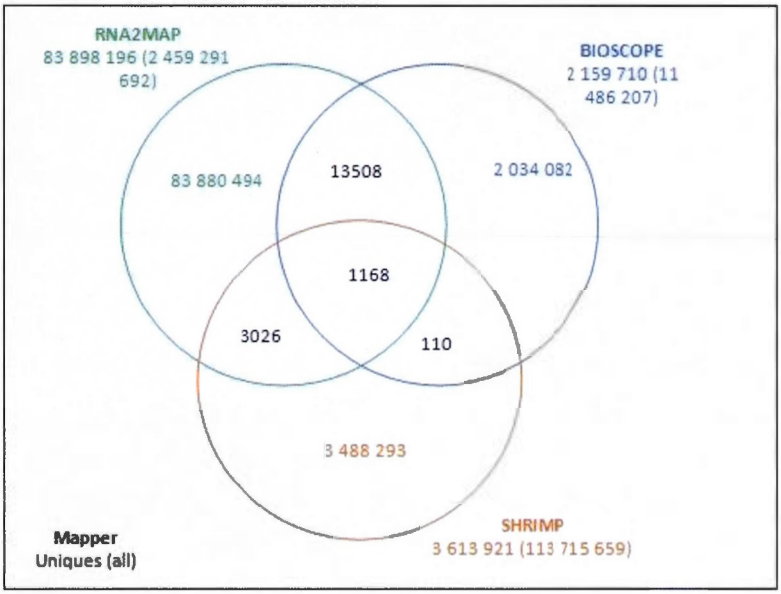


Figure 3.3 Comptage des fragments alignés lors d'essais de méthodes d'alignement

Finalement, après un alignement des fragments en séquences couleurs contre les ESTs du blé avec *MAQ*, nous avons obtenu 17 998 119 fragments (5 431 878 petits ARNs uniques) alignés. Ce qui donne un pourcentage d'alignement de 66,3% entre 16 et 31pb. Ce taux est au-dessus de la moyenne des taux rencontrés dans la littérature et des autres programmes d'alignement testés. Il important de noter ici, que sur tous les fragments alignés, seulement 9 483 473 fragments (342 433 petits ARNs uniques), avaient une expression totale supérieure à 5 dans l'ensemble des librairies.

Après l'alignement des fragments, sur l'ensemble des librairies exposées dans la Figure 3.4, quatre pics sont observables aux tailles 16, 17 et 19. La librairie 3 montre un schéma différent, expliqué par le nombre moins élevé de fragments après le séquençage par rapport à toutes les autres librairies. Aussi, les librairies 7, 8, 9 et 10 ont des abondances plus grandes. Les graphiques sont relativement similaires entre les librairies car la plupart des microARNs sont souvent exprimés au moins une fois dans chaque librairie. Les pics observés après la suppression des adaptateurs dans la Figure 3.2A, aux tailles 21 et 24 sont presque effacés

dans ces distributions. Les petits ARNs ont donc été alignés en plus grande quantité par rapport aux plus longs d'entre eux. Ceci peut s'expliquer par le fait que seulement 66.3% des petits ARNs ont été alignés sur les ESTs, donc une grande part d'entre eux ne sont pas identifiables. Toutefois, cette forte proportion de petits ARNs fini par diminuer au cours des étapes qui suivent l'alignement. En effet, nous pouvons le constater à la fin du processus de prédiction dans la Figure 3.5. Notons que dans cette figure, l'échelle diffère puisque nous comparons l'étape après alignement et après prédiction. Cette figure représente l'abondance par taille des microARNs dont le précurseur a été prédit par les deux logiciels de prédiction HHMMiR et miPred ; et dont la position sur le précurseur a été validée par *miRdup*. Ici, les distributions des tailles par librairies montrent un schéma relativement similaire entre elles. Seule la librairie 3 possède moins de microARNs de taille 16. Les pics des microARNs de 21 et 24 nucléotides sont plus visibles. Ceci indique que la plupart des petits ARNs n'étaient pas de microARNs puisqu'ils n'ont pas été prédits ainsi. Une part tout de même élevée de petits microARNs subsiste, ce qui diffère de la distribution générale des microARNs de miRbase (voir Figure 3.6). Tous les microARNs n'ont pu être identifiés dans le blé par cette étude. Puisque nous n'avons pas le génome complet, nous ne pouvons pas avancer que la plupart des microARNs du blé ont majoritairement une taille en dessous de 20 nt.

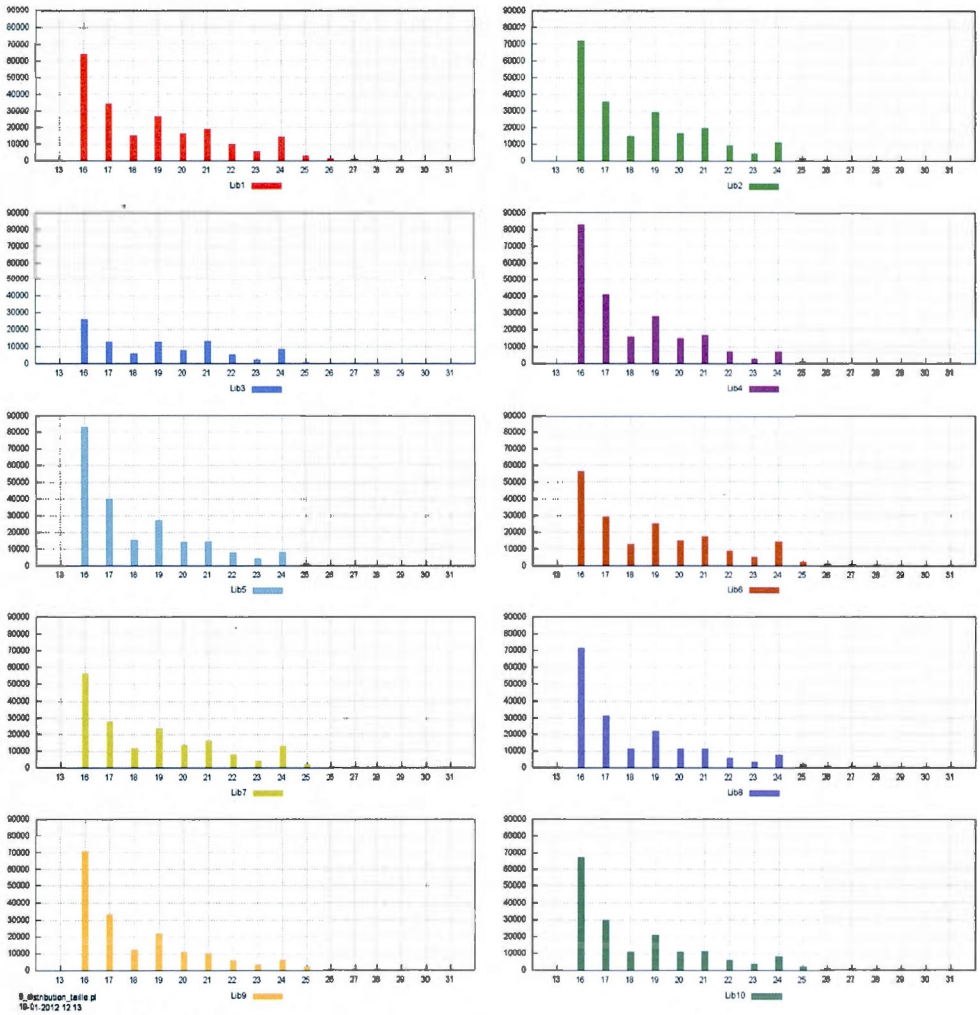


Figure 3.4 Distributions des tailles des petits ARNs uniques alignés après l'alignement

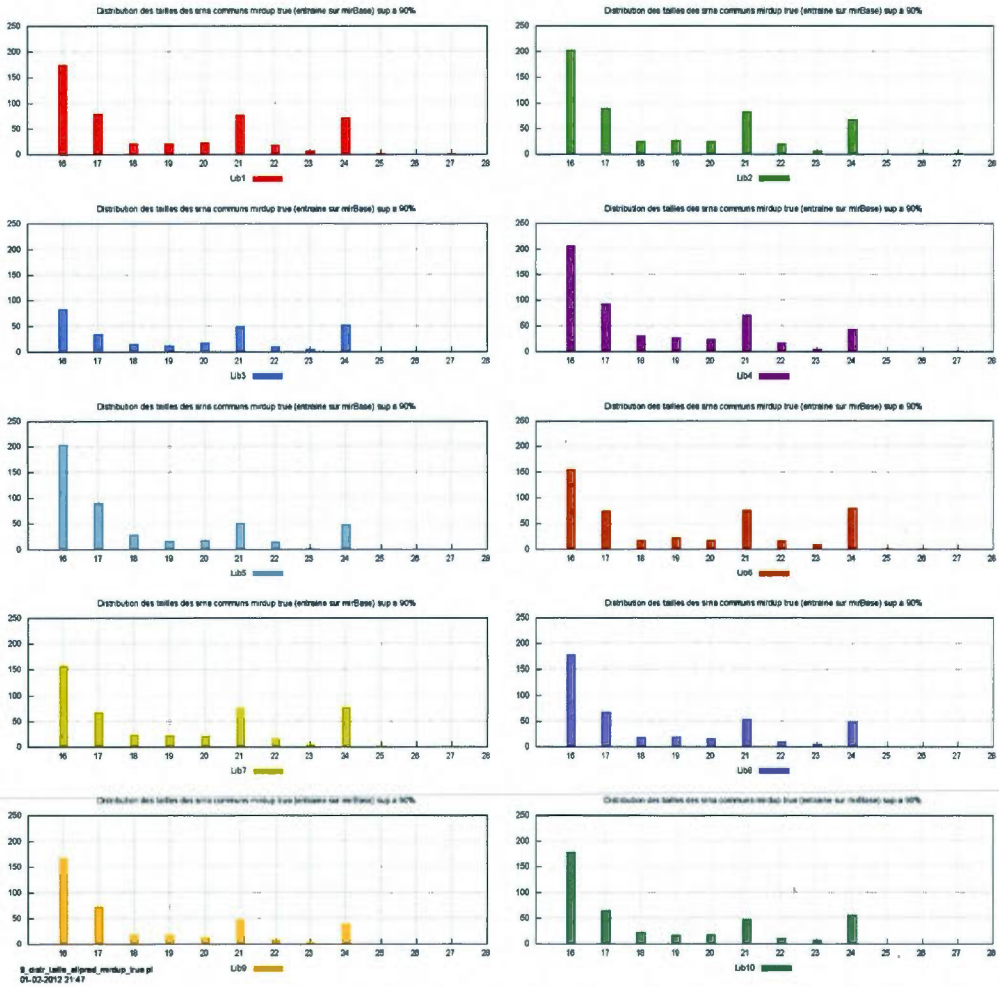


Figure 3.5 Distributions des tailles des microARNs uniques prédits en communs et validés par *mirDup*

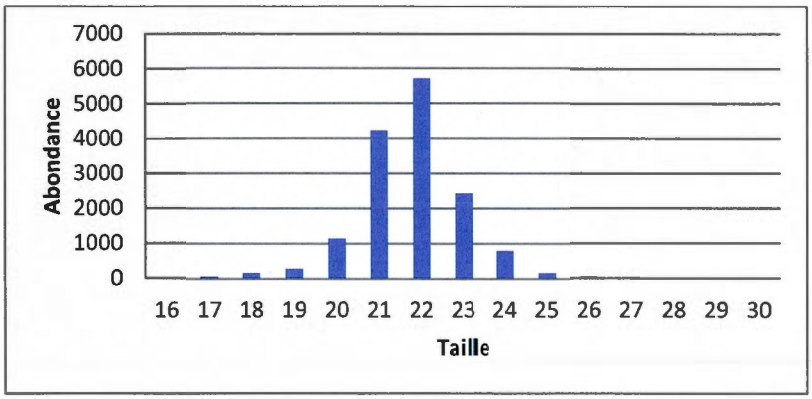


Figure 3.6 Distribution des tailles des microARNs de miRbase

3.2.3.2 *MicroARNs prédits*

Les précurseurs des séquences obtenues par alignement ont été extraits des ESTs puis soumis au repliement avec le programme *RNAfold* et aux logiciels de prédiction *HHMMiR* et *miPred*. En premier, un test positif des logiciels de prédiction choisis a été effectué sur les précurseurs de miRbase v17 et PMRD après le alignement. Environ 83% des précurseurs de miRbase, dont 12/44 du blé, et 62% de PMRD (44/85 blé) ont été prédits comme des pré-miRNAs potentiels par *miPred*, et 78% de miRbase (28/44 blé) et 69% de PMRD (58/85 blé) par *HHMMiR*. Ainsi, il apparaît qu'environ 20% des précurseurs risquent de ne pas être prédits avec cette approche. Cependant, cette rigueur dans les critères qui éliminent ces précurseurs permet de réduire de façon drastique le taux de faux positifs. Pour HHMMiR, la limite du ratio entre la vraisemblance des microARNs et les faux microARNs a été calculée à 0.995.

Un total de 98 422 précurseurs uniques portant 96 671 microARNs uniques ont été prédits, dont 27 957 (28 141 microARNs) par miPred, et 71 983 (81 376 microARNs) par *HHMMiR*. En retenant les résultats communs aux deux logiciels de prédiction, 915 précurseurs pour 11 073 microARNs ont été obtenus. Ensuite, 4 020 microARNs potentiels ont été retirés à cause d'une absence de gènes cibles, 3 418 pour une grande similarité avec des gènes ribosomaux, 302 à cause de la similarité aux ARNs non codants et 60 à cause d'une faible complexité. Ainsi, 3 862 microARNs portés par 7 036 précurseurs ont passé tous les tests par notre

pipeline d'analyse. Ces derniers ont été catégorisés en fonction de leur valeur de pvalue, leur résultat de classification, leur type d'épingle à cheveu (réelle ou pseudo), provenant de miPred afin de mieux visualiser les meilleurs d'entre eux. Quatre catégories ont donc ainsi été créées : A: pvalue<0.05 et réelle; B : pvalue<0.05 et pseudo; C: pvalue>0.05 et réelle ; D: pvalue>0.05 et pseudo. La catégorie A représente la meilleure catégorie. La répartition des microARNs dans les différentes catégories est présentée dans le Tableau 3.4, où l'ont obtient 800 microARNs dans la catégorie A. La majorité d'entre eux se trouve dans la catégorie D. Le compte total totalise plus que 3862 microARNs puisque certains d'entre eux ont plusieurs précurseurs qui peuvent se retrouver dans différentes catégories. De même, un même précurseur peut se retrouver dans différentes catégories puisqu'il y a un facteur aléatoire dans le calcul de la pvalue. Dans ce dernier cas un tel précurseur peut appartenir aux catégories A et B ou C et D.

Tableau 3.4 Répartition des microARNs prédits dans les différentes catégories

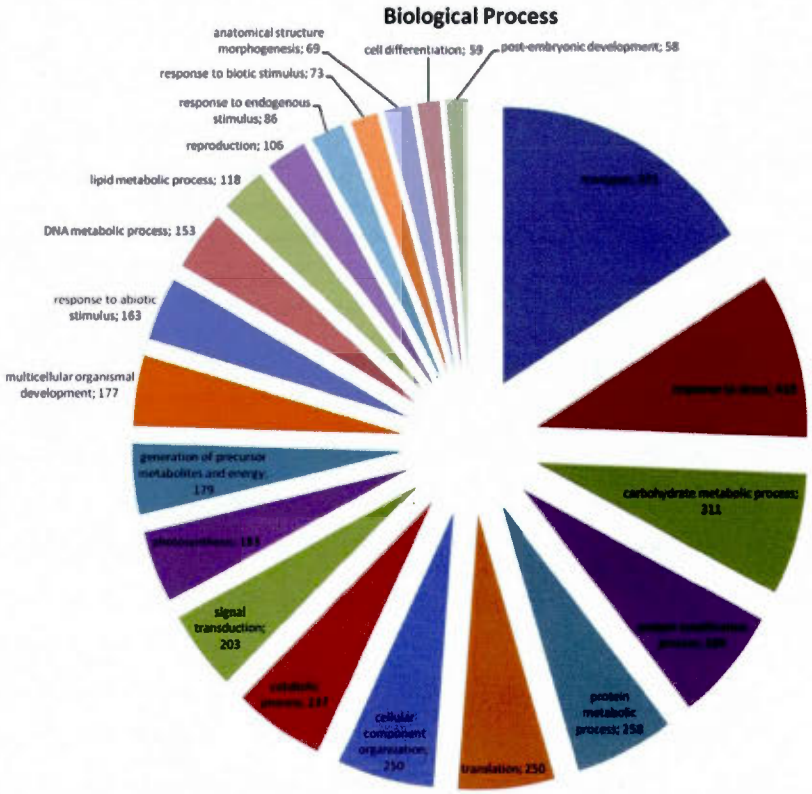
Catégorie	Nombre de microARNs	Nombre de précurseurs	Paramètres
A	806	1398	pvalue<0.05 et réelle
B	307	584	pvalue<0.05 et pseudo
C	5	9	pvalue>0.05 et réelle
D	2940	5226	pvalue>0.05 et pseudo

Les résultats obtenus montrent qu'un microARN peut avoir plusieurs précurseurs à cause de la fusion des résultats de *HHMMiR* et *miPred*, car ces derniers peuvent sélectionner des parties différentes du précurseur. Les résultats montrent aussi que plusieurs fragments différents peuvent produire exactement la même séquence nucléotidique. Ceci s'explique par la séquence en couleur, la position de l'adaptateur, l'existence de mésappariements et les variations de la qualité du fragment initial. En plus, une même séquence couleur, avec une qualité différente, peut aussi s'aligner sur des endroits différents et avoir une séquence légèrement dissemblable (voir chapitre II pour plus de détails). Pour pallier à ces redondances qui peuvent affecter l'analyse des profils d'expression, nous avons décidé d'additionner les expressions des répétitions où plusieurs fragments différents pouvaient donner la même séquence précurseur.

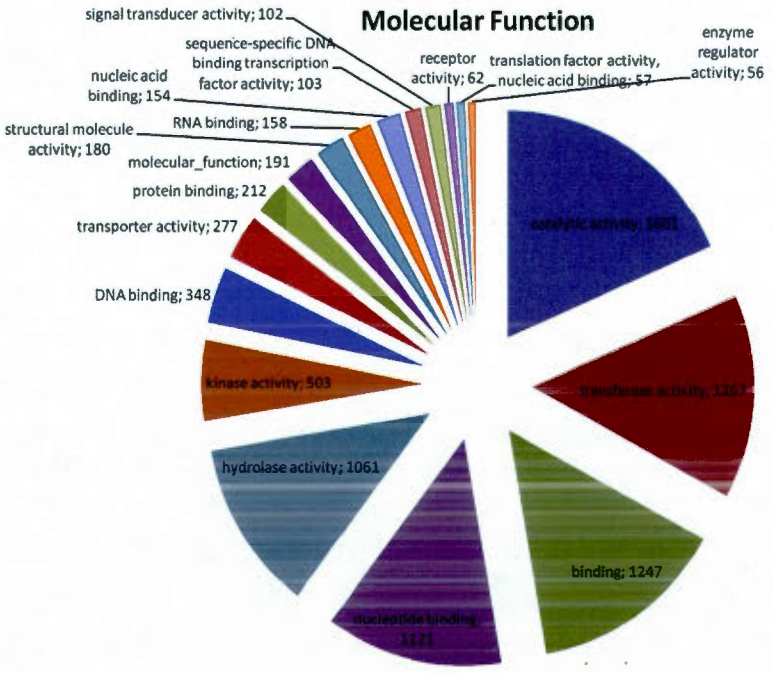
3.2.3.3 *Gènes ciblés par les microARNs prédits et leur fonction*

Afin d'identifier les gènes cibles des 3 862 microARNs prédits, nous avons utilisé le programme TAPIR. Au total, 71 891 ESTs ont été ciblés par les microARNs. Le nombre d'ESTs ciblés par un microARN varie de 1 à plus de 8 262 gènes ciblés. La moyenne de gènes ciblés par un microARN est de 25 gènes. En effet, 14 d'entre eux ont plus de 1 000 cibles. Parmi ces cibles, 5 303 ont un numéro d'accession protéique correspondant à une entrée de la base de données UniProt et 10 392 d'entre eux ont juste des mots clés dans leur annotation. Le reste n'a pas d'annotation. Les résultats bruts montrent que les gènes ciblés ont des activités très variées. Les gènes cibles possédant une référence protéique sont dispersés dans les trois grandes catégories de l'ontogie de Gènes GO: processus biologiques, fonctions moléculaires et composants cellulaires. Ces trois catégories sont elles même déclinées en dizaines de sous catégories permettant de classer des gènes selon leur fonction. La liste des références protéiques a été classée par GORetriever, dont les résultats sont présentés dans la Figure 3.7. Ces trois grandes catégories sont séparées en trois graphiques incluant les sous catégories dans lesquelles sont classés les 5303 gènes. Afin de montrer les plus représentatives, seule une portion des sous catégories est présentée ici. Les classes contenant moins de 50 gènes classés ne sont pas affichées. Ces figures montrent que les microARNs ciblent des gènes qui sont associés à des processus cellulaires, métaboliques et structuraux très variés. Par exemple, dans la Figure 3.7A, 691 gènes ciblés sont impliqués dans les processus de transport et 419 à la réponse au stress. Dans Figure 3.7B, 1 601 gènes sont impliqués dans des activités catalytiques, 1 267 dans des activités transférases et 1 247 dans des activités de liaison. Enfin, dans la Figure 3.7C, 678 gènes sont impliqués dans des processus localisés dans les plastides, 662 dans le cytoplasme, 352 dans le noyau, etc.

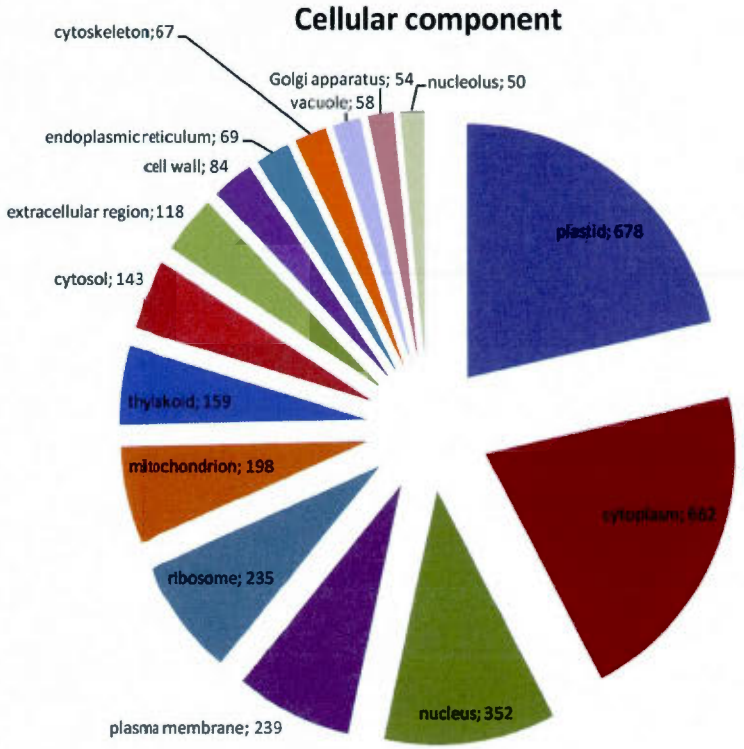
Dans cette étude, la caractérisation des gènes cibles se combine à l'expression différentielle. En effet, l'analyse de l'expression différentielle, décrite plus loin, permet d'ajouter une fonction globale quant à l'implication de ces microARNs prédits dans les processus de développement, de réponse et tolérance aux stress, et à leur localisation dans les feuilles ou les racines.


Biological Process
anatomical structure morphogenesis; 69
cell differentiation; 59
post-embryonic development; 58
response to biotic stimulus; 73
response to endogenous stimulus; 86
reproduction; 106
lipid metabolic process; 118
DNA metabolic process; 153
response to abiotic stimulus; 163
multicellular organismal development; 177
generation of precursor metabolites and energy; 179
carbohydrate metabolic process; 311
signal transduction; 203
cellular component organization; 250
translation; 250
protein metabolic process; 258

A


Molecular Function
signal transducer activity; 102
sequence-specific DNA binding transcription factor activity; 103
receptor activity; 62
translation factor activity, nucleic acid binding; 57
enzyme regulator activity; 56
nucleic acid binding; 154
RNA binding; 158
structural molecule activity; 180
molecular_function; 191
protein binding; 212
transporter activity; 277
DNA binding; 348
kinase activity; 503
hydrolase activity; 1061
binding; 1247

B



C

Figure 3.7 Ontologies des gènes ciblés par les microARNs prédits

3.2.3.4 *Validation des prédictions basées sur le duplexe miRNA-miRNA**

Afin d'améliorer les résultats de prédiction, nous avons décidé de modéliser et d'entrainer un modèle de validation en exploitant la puissance des techniques d'apprentissage machine. La première étape consiste à choisir les caractéristiques, ou attributs, qui définissent notre modèle. Afin de rendre ce modèle efficace, 35 attributs ont été identifiés en fonction de leur association aux petites séquences de microARN sur son épingle à cheveux. Parmi ceux-ci, quelques-uns sont basés sur la position et la longueur de plusieurs éléments structuraux dans le précurseur du microARN, comme les hernies et les paires de bases, ou sur la position du microARN en lui-même sur la structure (par exemple la distance relative à la boucle terminale). La Figure 3.8A présente ces exemples, où quelques attributs du duplexe

microARN et sa séquence complémentaire sont affichés dans le cadre bleu. Aussi, dans nos résultats il arrive qu'un microARN s'étende dans la boucle (Figure 3.8B) ou qu'un des nucléotides, A, G, U ou C, ne soit pas présent dans la séquence. À la Figure 3.8B, nous montrons qu'il peut arriver qu'une séquence prédite comme un précurseur pouvant porter un microARN, en noir sur la figure, par un programme de prédiction tel que *HHMMiR* ou *miPred*, le microARN séquencé peut s'étendre dans la boucle ou dans une grosse hernie. Dans cet exemple, si une épingle à cheveux est prédite comme vraie et que le microARN, en jaune, présente la possibilité d'une bonne position, la méthode lui attribuera un meilleur score de prédiction. À l'inverse, pour d'autres microARNs, comme ceux en rouge, une partie de ces derniers est située dans la boucle ou dans une grosse hernie, affectant alors le processus biologique de la protéine *Dicer*. Conséquemment, il est important que leur score de prédiction soit moindre. Enfin, quelques statistiques ont été calculées, tels que le pourcentage de la présence de certains nucléotides et de GC ainsi que l'énergie de repliement minimum du duplex microARN-microARN*. Un dernier attribut est la classe, qui est fixée vraie ou fausse dépendamment des données positives ou négatives respectivement. La liste des attributs complète est présentée en annexe au point *miRdup*.

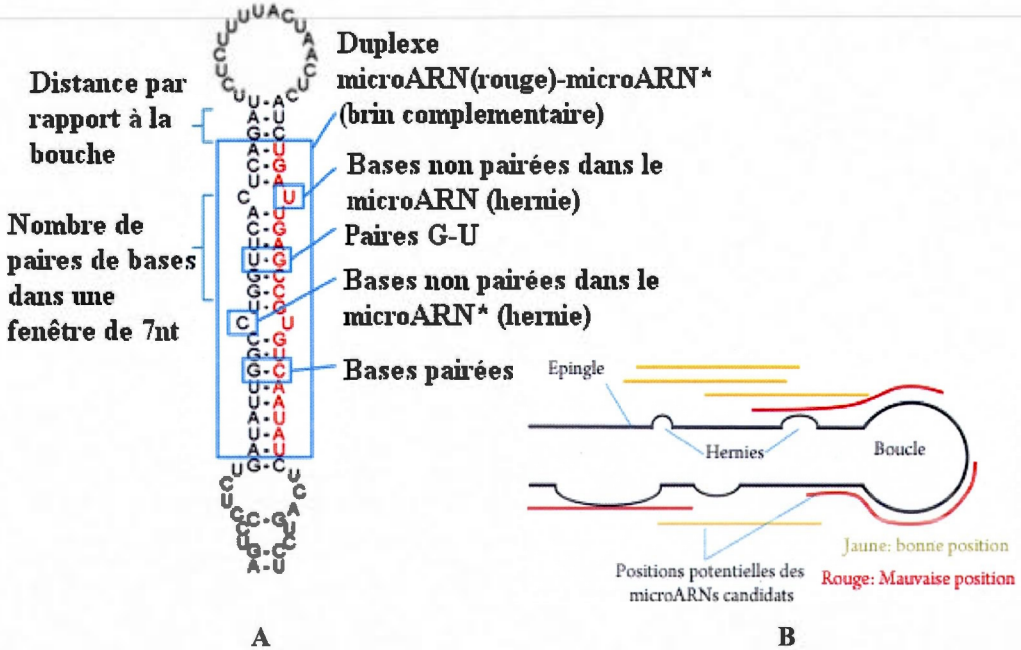


Figure 3.8 Attributs et positions du duplexe microARN-microARN*

La sélection des attributs a été effectuée de manière à réduire ceux qui n'ont aucune influence sur le modèle. Seuls 14 caractéristiques sont utilisées pour entraîner le modèle (Tableau 3.5). Le temps de calcul pour la sélection des caractéristiques est très rapide.

Tableau 3.5 Attributs utilisés dans le modèle de classification avec leur score de rang

Attributs	Score
Distance à partir de la tête de l'épingle (boucle)	0.2260145
Longueur du chevauchement sur la boucle	0.2110052
Nombre moyen de paires de bases dans une fenêtre de 3 nucléotides	0.1999772
Nombre de paires de bases dans le duplexe	0.1870463
Taille de la plus grande hernie en pourcentage des bases du duplexe	0.1858766
Nombre moyen de paires de bases dans une fenêtre de 5 nucléotides	0.1843222
Longueur de la plus grande hernie	0.1820702
Nombre moyen de paires de bases dans une fenêtre de 7 nucléotides	0.1664599
MicroARN inclus dans la boucle	0.1066618
Distance du microARN à partir du début de l'épingle	0.0865451
[illegible]	[illegible]
Longueur maximale sans hernie	0.070356
Longueur maximale sans hernie en pourcentage des bases du duplexe	0.069198
MFE du duplexe	0.0551582

Une fois les attributs définis, la deuxième étape consiste à entrainer le modèle de validation sur différents algorithmes, appelés *classificateurs*. Plusieurs d'entre eux, parmi les plus employés dans la littérature, ont été testés dans le but de trouver le meilleur qui pourrait discriminer efficacement les données en fonction de la classe (Tableau 3.6) où les meilleurs scores par colonnes sont affichés en vert et les plus bas scores en rouge. Les données sont ordonnées selon le meilleur taux de faux positif, c'est-à-dire le plus faible. Tous on été entrainés en effectuant la validation croisée 10 fois. Nous considérons qu'un classificateur est meilleur qu'un autre en comparant les instances correctement classées lors de l'apprentissage, et sa faculté à détecter les faux positifs.

Une machine à vecteur de supports, SVM, a été testée avec une base de cœur radial et linéaire. Les deux offrent des résultats très différents avec plus de 5% d'écart sur les instances correctement classées et de larges disparités entre les taux de vrais/faux positifs/négatifs. Cependant le SVM avec un cœur radial donne globalement les meilleurs résultats, comme on peut le voir dans le Tableau 3.6. Une limite au SVM est qu'il est très long à entrainer, de l'ordre de plusieurs heures par rapport aux autres classificateurs. D'un autre coté, des résultats intéressants apparaissent sur un arbre de décision C4.5 (J48) avec de la stimulation par la méthode Adaboost M1. Plusieurs autres méthodes de classification ont été essayées, comme un réseau de neurones multicouche avec rétropropagation, la régression logistique multinominale et SMO (Sequential minimal optimisation en anglais), qui est un algorithme proche du SVM.

Tableau 3.6 Résultats obtenus par les classifieurs testés

Classifieur	Paramètres	Temps pour construire le modèle (secondes)	Instances correctement classées (%)	Taux de vrais positifs	Taux de faux positifs	Taux de vrais négatifs	Taux de faux négatifs	Sensitivité	Spécificité
SVM coeur radial	-S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.0010 -P 0.1	1417.79	77.5184	0.734	0.185	0.815	0.266	0.734	0.815
TreeJ48 avec Adaboost	AdaBoostM1 -P 100 -S 1 -I 10 -W trees.J48 -C 0.25 -M 2	168.09	76.9488	0.789	0.25	0.75	0.211	0,789	0,75
TreeJ48	-C 0.25 -M 2	13.94	76.66	0.813	0.279	0.721	0.187	0,813	0,721
JRip	-F 10 -N 2.0 -O 2 -S 1	123.83	78.467	0.882	0.31	0.69	0.118	0,882	0,69
Perceptron multicouches	-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a	164.55	75.5869	0.851	0.337	0.663	0.149	0,851	0,663
Régression logistique	-R 1.0E-8 -M -1	2.5	73.6447	0.868	0.392	0.608	0.132	0,868	0,608
SMO	-C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K PolyKernel -C 250007 -E 1.0	40.27	73.3268	0.921	0.45	0.55	0.079	0,921	0,55
SVM, coeur linéaire	-S 0 -K 0 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0 -C 1.0 -E 0.0010 -P 0.1	6561.18	72.2511	0.962	0.51	0.49	0.038	0.962	0.49

La Figure 3.9 présente les courbes de performance (*ROC curves* en anglais) des taux de vrais positifs comparés aux faux positifs pour chaque type de classificateurs. Dans le cas d'un classificateur qui n'a pas de classe de probabilités propre, comme le SVM, il n'y a que deux points sur le graphe. Nous observons qu'Adaboost sur un arbre de décision surpasse tous les autres classificateurs. Cependant, sur une petite portion le SVM avec un cœur radial, entourée sur la figure, montre qu'il est le plus efficace sur un taux précis.

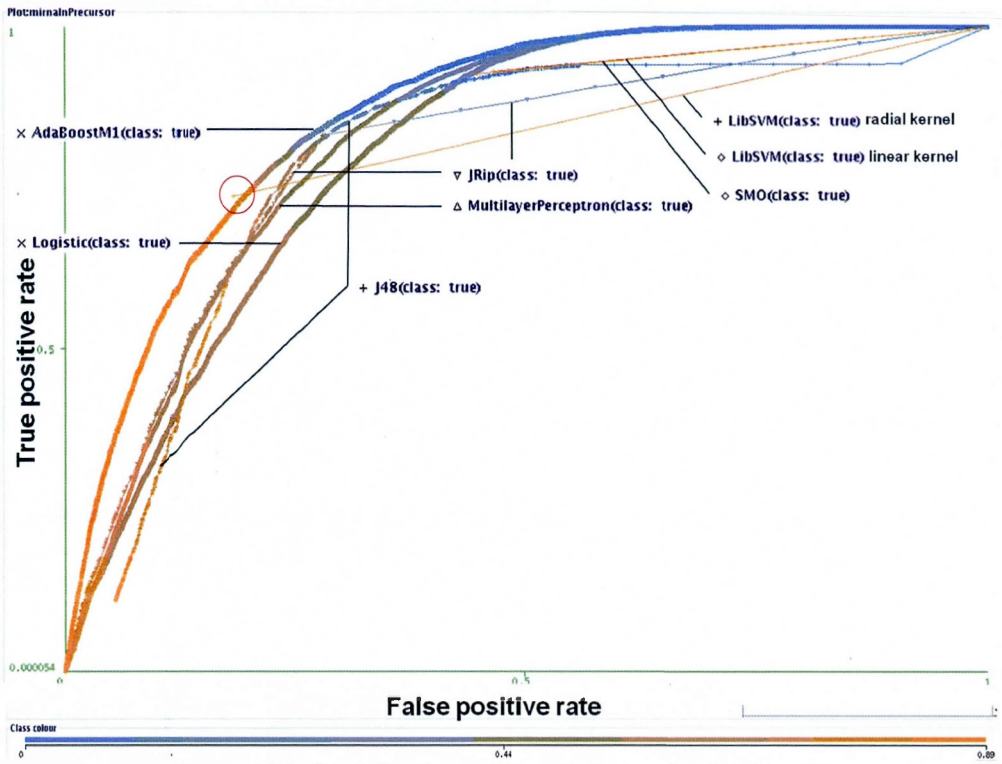


Figure 3.9 Courbes de performances des classificateurs

Après l'entrainement des différents classificateurs sur le jeu de données d'entrainement, un nouveau jeu de données, appelé jeu de données contrôle, a été soumis à chacun des modèles construits. Ce jeu provient de PMRD. Le Tableau 3.7 présente les résultats des classificateurs entrainés dans le Tableau 3.6, où le même principe de couleurs est employé. Des résultats similaires sont retrouvés entre les deux tables, avec des hauts taux d'instances correctement classées. Tout comme durant la phase d'entrainement, nous retrouvons un ordre similaire entre les classificateurs. Le SVM avec un cœur de base radiale et l'arbre de décision

avec Adaboost ont les meilleurs scores sur tous les points. Ils ont correctement classé plus de 89% et 85% respectivement, en préservant un très faible taux de faux positifs.

Tableau 3.7 Résultats d'évaluation des classificateurs sur le jeu de données de contrôle

Classifieur	Instances correctement classées (%)	Taux de vrais positifs	Taux de faux positifs	Taux de vrais négatifs	Taux de faux négatifs	Sensitivité	Spécificité
SVM coeur radial	89,4827	0,986	0,197	0,803	0,014	0,986	0,803
TreeJ48 avec Adaboost	85,679	0,991	0,278	0,722	0,009	0,991	0,722
TreeJ48	80,2645	0,923	0,318	0,682	0,077	0,923	0,682
Perceptron multicouches	75,8588	0,856	0,338	0,662	0,144	0,856	0,662
Jrip	76,297	0,897	0,371	0,629	0,103	0,897	0,629
Régression logistique	74,3016	0,919	0,433	0,567	0,081	0,919	0,567
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
SVM coeur linéaire	70,8819	0,981	0,564	0,436	0,019	0,981	0,436

Par la suite, les 8 678 microARNs et leurs précurseurs prédits ont été soumis aux modèles entrainés de la

. Dans ce jeu de données, plusieurs microARNs identiques ont des précurseurs différents et plusieurs précurseurs identiques ont des microARNs différents, puisque dans ce lot il y a 3 862 microARNs pour 7 036 précurseurs uniques. Dans le Tableau 3.8, le SVM avec un cœur de base radiale montre qu'environ 3.8% de nos prédictions sont considérées comme des microARNs soutenus par les précurseurs donnés. À l'inverse, si l'on utilise celui avec un cœur linéaire, 32% sont considérés comme bons. Mais considérant les résultats obtenus avec le jeu de données d'entrainement et de contrôle, nous ne choisirons pas le SVM avec un cœur linéaire à cause de ses mauvais scores de classement. D'un autre point de vue, 3.8% de bonnes prédictions est beaucoup trop restrictif. Ainsi, une option rationnelle consiste à utiliser un classificateur intermédiaire comme l'arbre de décision avec Adaboost, qui montre 19.28% de bonnes prédictions dans nos microARNs.

Tableau 3.8 Évaluation des prédictions par les classificateurs

Modèle	Prédits correctement (%)
SVM cœur radial	3.7912
Perceptron multicouches	11.1662
Régression logistique	17.9419
TreeJ48 avec Adaboost	19.2786
JRip	20.0161
SMO	20.5232
TreeJ48	26.5268
SVM cœur linéaire	32.0581

En conclusion, après avoir entrainé plusieurs classificateurs, nous avons retenu : le SVM avec un coeur de base radiale et Adaboost sur un arbre de décision. Le SVM se montre toutefois très strict sur les résultats de notre jeu de microARNs prédits par rapport à l'arbre de décision avec Adaboost. Ce dernier a une très bonne spécificité, donc une bonne habilité à discriminer les résultats négatifs. C'est aussi celui qui offre le meilleur compromis entre les scores de classification et les évaluations des prédictions de l'étude.

En prenant les microARNs prédits en communs par les deux logiciels de prédiction de précurseurs et la validation par *miRdup* avec un arbre de décision, 1 673 instances de microARNs avec leur précurseur sont validées par notre processus. Ceci inclus 1 016 microARNs unique pour 1 499 précurseurs uniques. Toutefois, nous choisissons de simplement leur attribuer un meilleur score, de façon à ne pas rejeter les autres prédictions.

3.2.3.5 *Composition des fragments en classes d'ARN*

Cette étape a été effectuée à deux moments, après l'alignement et après les prédictions. Lors de l'analyse de la composition des fragments en ARNs non codants après l'étape d'alignement sur les ESTs, nos résultats ont montré que plusieurs types y sont présents. Dans des proportions marginales, entre quelques dizaines à quelques centaines, ont été détecté comme (Air), ERalpha (Eralpha antisense), KLHL1 (Kelch-like 1 antisense), RNase (Ribonuclease), SRP (Signal recognition particule), XIST (X chromosome inactivation),

mRNAlike (Messenger), piRNA (Piwi-interacting), self-splicing (self-splicing ribozyme), snRNA (Small nuclear), snmRNA (Small non-messenger), snoRNA (Small nucleolar), tmRNA (Transfer-messenger) et d'autres encore. Ce résultat de *BLAST*, constitue un moyen efficace d'éliminer ces produits de dégradation dus aux manipulations. Par contre des résultats montrent la présence d'ARN piwi, or ces ARNs sont connus seulement chez les vertébrés (Seto et al. 2007). Pour expliquer ce résultat, il faut tenir compte que même si un petit ARN aligné est trouvé sans aucun mésappariement, ils ne sont similaires que sur une petite partie de la région des non codants. Une grande partie de ces non codants détectés sont donc probablement de faux positifs. Néanmoins il n'est peut être pas exclu que cette affirmation soit fausse et qu'il en existe chez les plantes aussi. Par ailleurs, très peu d'ARNs ribosomaux ont été trouvés sur RNA database v2.0. Ainsi, nous avons comparé nos données par *BLAST* contre les bases de données du NCBI pour chercher plus de ribosomaux. Toutefois, à cause de la lourdeur computationnelle de cette étape, seuls les microARNs prédits ont été comparés.

3.2.3.6 *MicroARNs conservés parmi les fragments*

Par ailleurs, afin de savoir si des microARNs publiés dans miRbase étaient présents dans nos données de séquençage, un *BLAST* a été effectué contre miRbase. Au total, 1758 petits ARNs séquencés sont retrouvés dans miRbase, dont 37 chez le blé, sachant que 41 sont publiés. La Figure 3.10 présente l'arbre phylogénétique de la conservation des fragments après suppression des adaptateurs et alignement dans miRbase. L'arbre provient d'une instance de l'arbre de la vie obtenu à partir d'*iTol* (Letunic & Bork 2007). Les espèces monocotylédons sont en vert, les dicotylédons en bleu, les chordés en beige et Picea en violet. La plupart des espèces ne sont pas représentées puisqu'aucun microARN n'est conservé avec le blé. Les barres rouges externes représentent les correspondances (*hits*) de blasts contre une espèce spécifique. Les barres vertes sont le pourcentage de ces correspondances sur le nombre de microARNs dans chaque espèce. Seules les espèces où des microARNs ont été retrouvés dans nos données sont représentées ici. Nous observons une forte conservation chez les monocotylédons, incluant le blé, et chez quelques dicotylédones. Toutefois, les résultats ne montrent pas de correspondance entre la distance évolutive dans l'arbre et le nombre de microARNs conservés entre une espèce donnée et le

blé. Ceci peut s'expliquer par la qualité du séquençage, les conditions biologiques dans lesquelles ont été soumises les plantes, la sélection de lignée incomplète ou encore la partialité des bases de données. Sans surprise, la conservation chez les chordés est très faible, voire quasi inexistante. Nous expliquons leur présence plus loin.

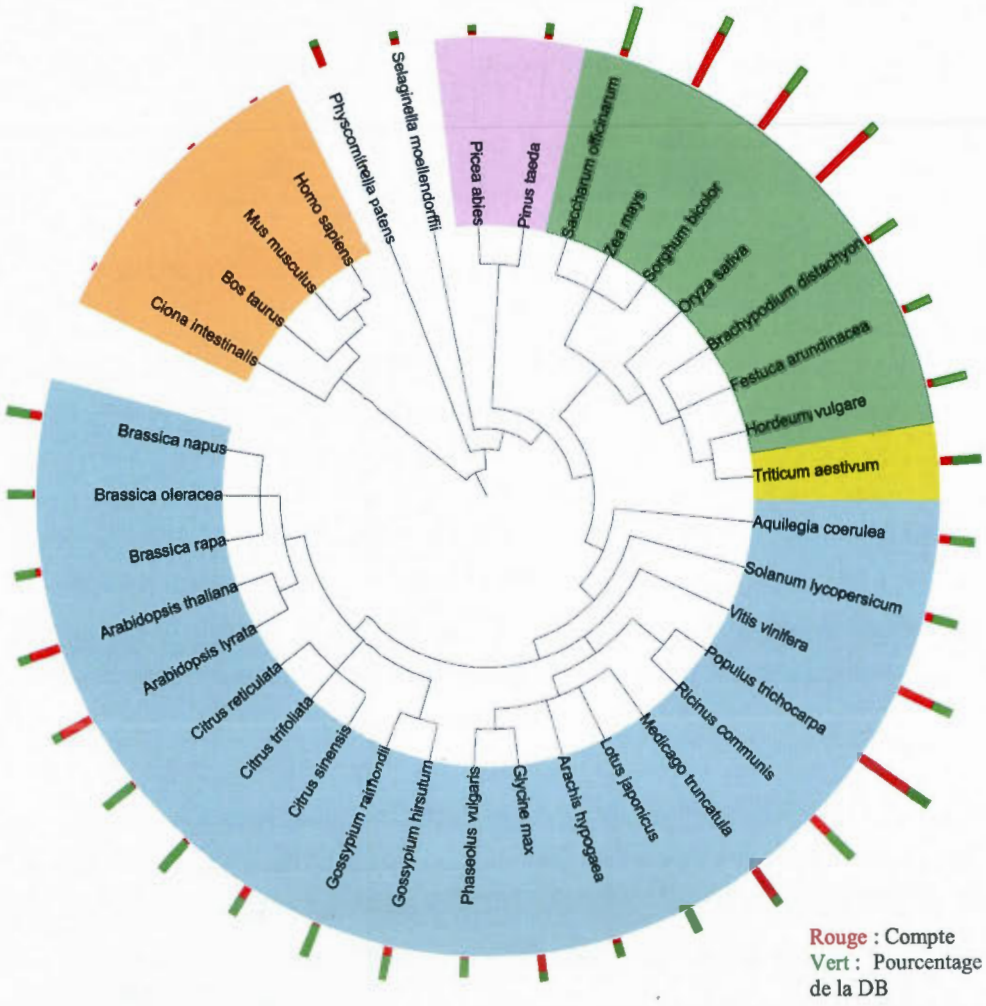


Figure 3.10 Arbre phylogénétique des microARNs conservés dans miRbase

À partir d'un *BLAST* contre la base de données de miRbase, nous avons pu identifier seulement 12 microARNs sur 3 862 qui sont conservés, dont 2 chez le blé (voir Tableau 3.9). Les autres espèces montrant une petite conservation sont *Oryza sativa* (osa), *Hordeum vulgare* (hvu) et *Ciona intestinalis* (cin).

Tableau 3.9 Résultats de la recherche d'homologie des microARNs dans miRbase

microARN prédit	microARN miRbase
GACUCUCUUAAGGUAGCC	cin-miR-4171-5p
ACUCUCUUAAGGUAGCCA	cin-miR-4171-5p
GACUCUCUUAAGGUAGCCAA	cin-miR-4171-5p
UGACUCUCUUAAGGUAGCCA	cin-miR-4171-5p
ACUCUCUUAAGGUAGCCAAA	cin-miR-4171-5p
AUAUUUUGGGACGGAGGUAGU	hvu-miR1436
AUAUUUUGGGACGGAGGUAGU	osa-miR1436
AUUUUGGGACGGAGGGAGUAA	hvu-miR1436
AUUUUGGGACGGAGGGAGUAA	osa-miR1436
AUUUUGGGACGGAGGGAGUAA	osa-miR1439
UCCGUCCCAUAAUGUAAGACG	tae-miR1130
UAUGACUCUCUUAAGGUAGCC	cin-miR-4171-5p
UGACUCUCUUAAGGUAGCCAA	cin-miR-4171-5p
UAUGACUCUCUUAAGGUAGCCA	cin-miR-4171-5p
UAGUCCCGGUUGGUGGCACGAACC	tae-miR1117

Le même processus a été effectué sur PMRD, où 4137 petit ARNs sont conservés, dont 71 sur 85 du blé. Après les prédictions, seuls 28 sont retrouvés conservés dans la base de données, dont 6 chez le blé et 17 chez le riz sur les 2780 publiés.

Après l'alignement, plus de 90% des microARNs publiés du blé sont présents dans nos données. Après les prédictions, seuls 5% le sont, ce qui s'explique par le fait que nous avons appliqué des paramètres très rigoureux dans l'ensemble de cette étude, notamment pour éviter l'abondance de faux positifs. Aussi, comme présenté dans les résultats, les logiciels de prédictions ne détectent pas non plus tous les précurseurs de miRbase ou PMRD comme vrais, certains ont donc été rejetés pour cette raison. De plus, le génome de référence composé d'ESTs portait probablement certains microARNs, mais leurs précurseurs n'étaient pas considérés comme vrais et ils n'ont pu être retenus. D'autre part, de nombreux microARNs présents dans les régions introniques et intergéniques ne sont pas présents dans les ESTs, donc de nombreux fragments (environ la moitié d'entre eux) n'ont pu être alignés. Finalement de nombreux microARNs ayant une taille très petite de 16pb ont vu leur précurseur prédit, alors qu'ils sont rares dans les bases de données. Nous pensons qu'il serait plutôt des faux positifs. Car à cause de leur petite taille, un biais aléatoire est présent. Ils ont

toutefois réussi à passer toutes les étapes de nos étapes d'analyse, donc peuvent potentiellement être des microARNs chez le blé.

Trouver un microARN de plante conservé chez les chordés peut s'expliquer en regardant les gènes qu'il cible grâce à *TAPIR*. Par exemple, le microARN UGACUCUCUUAAGGUAGCCA prédit est conservé chez *Ciona intestinalis*. Il cible à lui seul 39 ESTs, listés dans le Tableau 3.10. Parmi ces 39 cibles, certains ont des références qui permettent d'identifier les ontologies, telles que Q10P99, O04892 et A9U511. La protéine associée à O04892 est le cytochrome P450 régulant le Tributyl phosphate (Berne et al. 2007), qui a une activité d'oxydo-réduction et d'aromatase. Par ailleurs, la plupart des autres cibles ne portant pas de référence sont aussi des gènes codants pour P450. Les deux autres ontologies n'ont pas de fonction caractérisée. P450 est connu pour être présent chez de nombreuses espèces, animales ou plantes (Gonzalez & Nebert 1990), expliquant ainsi la conservation du microARN. Nous retrouvons aussi parmi les gènes ciblés par les microARNs, plusieurs gènes ribosomaux 25S et 26S. Enfin, la plupart de ces cibles ont un score maximal de complémentarité de 0, ce qui dénote une complémentarité parfaite, donc au final un clivage de l'ARNm.

Tableau 3.10 Cibles du microARN UGACUCUCUUAAGGUAGCCA

Accession EST	Annotation EST	Position ciblée	score
BJ218108	homologue to gb|AF036490.1| Acorus gramineus large subunit 26S ribosomal RNA gene partial sequence partial (9%)	216	0.0
BE590827	UPI00006A2900 UniRef100 entry [Xenopus tropicalis]	147	0.0
DT949014	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	352	0.0
BE515544	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	197	0.0
CJ516974	UPI00006A2900 UniRef100 entry [Xenopus tropicalis]	277	0.0
CJ508020	UPI00006A2900 UniRef100 entry [Xenopus tropicalis]	298	0.0
CA660571	UPI00006A2900 UniRef100 entry [Xenopus tropicalis]	290	0.0
BE443717	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	431	0.0
BE516835	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	396	0.0
BJ208110	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	353	0.0
BF484695	UPI00006A2900 UniRef100 entry [Xenopus tropicalis]	287	0.0
EB513481	UPI00006A2900 UniRef100 entry [Xenopus tropicalis]	163	0.0
BJ248866	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	387	0.0
BJ311277	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	497	0.0
BE517590	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	267	0.0

TC403896	homologue to UniRef100_Q10P99 Cluster: Senescence-associated protein expressed n=1 Oryza sativa Japonica Group|Rep: Senescence-associated protein expressed - Oryza sativa subsp. japonica (Rice) partial (21%)	56	0.0
BJ288872	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	450	0.0
CJ634618	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	303	0.0
BJ224645	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	535	0.0
FG618746	putative senescence-associated protein mRNA sequence.	503	0.0
BE516012	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	438	0.0
GD186866	similar to gb|AF036494.1|AF036494 Eucryphia lucida large subunit 26S ribosomal RNA gene partial sequence partial (18%)	484	0.0
BJ253165	Putative senescence-associated protein [Pisum sativum (Garden pea)]	579	0.0
GD187126	homologue to UniRef100_O04892 Cluster: Cytochrome P450 like_TBP n=1 Nicotiana tabacum|Rep: Cytochrome P450 like_TBP - Nicotiana tabacum (Common tobacco) partial (13%)	571	0.0
BJ303169	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	283	0.0
CK155165	homologue to UniRef100_O04892 Cluster: Cytochrome P450 like_TBP n=1 Nicotiana tabacum|Rep: Cytochrome P450 like_TBP - Nicotiana tabacum (Common tobacco) partial (10%)	533	0.0
CK164754	homologue to gb|AF036495.1|AF036495 Hamamelis virginiana large subunit 26S ribosomal RNA gene partial sequence partial (16%)	561	0.0
CK154773	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	572	0.0
CK205904	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	116	0.0
TA300_4571	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	637	0.0
TA53261_4565	Cytochrome P450 like_TBP [Nicotiana tabacum (Common tobacco)]	217	0.0
TC452799	similar to UniRef100_O04892 Cluster: Cytochrome P450 like_TBP n=1 Nicotiana tabacum|Rep: Cytochrome P450 like_TBP - Nicotiana tabacum (Common tobacco) partial (14%)	298	0.0
TC449482	homologue to gb|AF223066.1|AF223066 Humulus lupulus 26S ribosomal RNA gene partial sequence 18S ribosomal RNA gene internal transcribed spacer 1 5.8S ribosomal RNA gene internal transcribed spacer 2 and 26S ribosomal RNA gene complete sequence and 18S ribosomal RNA gene partial sequence partial (3%)	290	0.0
TC400716	similar to UniRef100_O04892 Cluster: Cytochrome P450 like_TBP n=1 Nicotiana tabacum|Rep: Cytochrome P450 like_TBP - Nicotiana tabacum (Common tobacco) partial (30%)	432	0.0
TA53289_4565	Putative senescence-associated protein [Pisum sativum (Garden pea)]	1322	0.0
CA657663	homologue to emb|X57137.1|SA25SR Sinapis alba gene for 25S rRNA partial (10%)	368	1.0
GD186476	similar to gb|AF036500.1|AF036500 Tellima grandiflora large subunit 26S ribosomal RNA gene partial sequence partial (15%)	597	1.0
GD186839	homologue to gb|AF036492.1|AF036492 Plumbago auriculata large subunit 26S ribosomal RNA gene partial sequence partial (16%)	466	2.0
DR733050	similar to UniRef100_A9U511 Cluster: Predicted protein n=1 Physcomitrella patens subsp. patens|Rep: Predicted protein - Physcomitrella patens subsp. patens partial (87%)	607	3.0

3.2.3.7 *Expression différentielle des microARNs*

Tous les microARNs prédits ne s'expriment pas en quantité égale entre les librairies. Certains d'entre eux possèdent de fortes variations d'abondance. L'analyse de l'expression dite différentielle de ces microARNs entre deux conditions opposées ou entre des plants tolérants

et sensibles à un stress permet d'y caractériser ceux qui y sont directement impliqués. Ainsi, en comparant une librairie provenant de blé tolérant et sensible à l'aluminium exposé à ce métal, des microARNs sur-exprimés et sous-exprimés dans une des plantes montrent qu'ils subissent une variation due à sa présence. Cette analyse peut être mise en liaison avec les gènes cibles, qui sont probablement impliqués dans la capacité de la plante tolérante à l'aluminium d'y résister.

Les analyses ont montré que les niveaux d'abondance des microARNs varient de 5 à plus de 12000 dans une librairie. Sur 3862 microARNs prédits, seuls 206 se sont révélés avoir une expression différentielle. Un microARN différentiellement exprimé entre deux conditions peut l'être dans plusieurs autres. Dans le Tableau 3.11, plusieurs librairies ont été comparées afin de rechercher les microARNs différentiellement exprimés entre chacune d'elles. Pour chaque comparaison, nous avons le nombre de microARNs sous-exprimés et sur-exprimés, c'est-à-dire dont l'abondance est inférieure ou supérieure respectivement entre deux librairies. Le nombre de microARNs ayant expression supérieure et inférieure à un total de 5 dans toutes les libraires sont affichés.

Tableau 3.11 microARNs différentiellement exprimés entre plusieurs conditions

Librairies	Conditions impliquées/Variant	Sous-expression	Sur-expression	Total	Fragments exp>5	Fragments exp<5
L2 vs L1	Réponse au froid/Clair	20	51	71	1558	2304
L4 vs L1	Réponse au sel/Clair	3	38	41	1584	2278
L7 vs L6	Réponse au froid/Bounty	9	34	43	1288	2574
L2 vs L7	Tolérance au froid	21	26	47	1420	2442
L3 vs L1	Réponse au développement	31	15	46	1066	2796
L10 vs L9	Réponse à l'aluminium/Atlas	28	24	52	1214	2648
L10 vs L8	Tolérance à l'aluminium	14	10	24	1161	2701
L8 vs L9	Réponse à l'aluminium/Bounty	20	23	43	1240	2622
L5 vs L4	Réponse au sel entre les racines et feuilles	34	32	66	1617	2245
Total		180	253	433	12148	22610

Le Tableau 3.12 présente le détail de la comparaison des librairies 8 et 10, qui représente la tolérance à l'aluminium. En effet, la librairie 8 est issue des tissus racinaires à partir de plants exposés à l'aluminium chez Bounty, une variété de blé sensible à l'aluminium. La librairie 10 est aussi issue de tissus racinaires à partir de plants exposés à l'aluminium, mais chez la variété Atlas, qui tolère la présence d'aluminium. En comparant ces deux librairies, nous pouvons ainsi caractériser les microARNs qui sont sur-exprimés dans la librairie 10 par rapport à librairie 8, ceux qui ont un z-score positif, ou sous-exprimés, avec un z-score négatif. Par exemple, au total, 10 microARNs sont sur-exprimés dans la librairie 10, et 14 sont sous-exprimés. En annexe C, le Tableau C1 contient la suite du Tableau 3.12 avec les ontologies de tous les gènes ciblés par ces microARNs différentiellement exprimés et portant une référence protéique.

Tableau 3.12 MicroARNs différentiellement exprimés entre les librairies 8 et 10

microARNs	Expression librairie 10	Expression librairie 8	Expression normalisée librairie 10	Expression normalisée librairie 8	z-score	p-value
UGGGGUGUAAUCUGCG	116	306	52	148	-2,86	4,94E-03
AAAAGCCAUCGACAAA	233	603	104	291	-2,8	7,64E-05
GGCCUACAUCGCUGAG	154	371	69	179	-2,61	3,69E-03
ACUGGUUGGAUCAUGCUUCUG	1530	2737	682	1320	-1,94	1,37E-09
UCCAAAUCGUGCUUUU	4762	8491	2121	4095	-1,93	5,35E-27
GGGCUUAGCUGCUUUU	190	335	85	162	-1,91	4,81E-02
UAUGUGCUAACAAAAA	1073	1795	478	866	-1,81	8,74E-06
UUUGAUCUAACGAAAA	2102	3386	936	1633	-1,74	5,73E-09
AAAAACACGAGAGUUU	828	1326	369	639	-1,73	3,85E-04
UUGUGGAGAUAAUGCAAACCC	2148	3307	957	1595	-1,67	9,42E-08
UUUGAGCUACGUUUUU	950	1460	423	704	-1,66	4,99E-04
UUUGCUAGCACAAAAA	831	1266	370	611	-1,65	1,47E-03
CCGUCUUCGUGUUUUU	863	1200	384	579	-1,51	1,04E-02
UAGUUCCUAGCUCAGA	3838	5094	1710	2456	-1,44	1,11E-06
GAAUGACAUCAGAUUC	611	289	272	139	1,95	2,93E-03
UCCGAGUCUGCUGCCU	549	260	245	125	1,95	4,74E-03
GAAUCUGAUGUCAUUC	610	288	272	139	1,96	2,90E-03
UCCUCUGCUCUACCAACUGAG.. ..CUAUCCUGAC	467	194	208	94	2,22	2,78E-03
AGGCAGCAUCUCUAGAU	136	38	61	18	3,31	2,81E-02
UCCUGAUGCUGAACUU	734	153	327	74	4,43	2,60E-08

AGGCAGCAUCUCUAGA	6543	882	2915	425	6,85	8,17E-77
AGGGUAUCGUGGCCAG	334	39	149	19	7,91	9,03E-06
UUCCUGAGCAUCAUUC	1032	64	460	31	14,9	4,50E-17
UCAAUACAUAUAUGACAA	190	10	85	5	17,55	1,78E-04

3.2.3.8 *Ensemble des informations extraites pour un fragment*

Dans les chapitres II et III, le fragment 1557_1102_758 est utilisé en exemple pour présenter l'évolution des résultats tout au long de l'étude. Au final, le Tableau 3.13 regroupe toutes les informations que nous avons obtenues pour ce fragment. Elle fait référence au contenu du Tableau 3.2. Pour chacun des fragments prédits les mêmes informations sont accessibles, mais nous n'afficherons pas les données dans ce mémoire pour des raisons d'espace.

Tableau 3.13 Informations obtenues dans l'étude pour le fragment 1557_1102_758

Information	Détails
Abondance du microARN*	9
Annotation de l'EST	Inconnu, pas d'annotation
Bras	3'
Brin	+
Catégorie	D
Commun	Oui
Conservé	Non
Couleur	T032022313123021010103233020103031 31
Différentiellement exprimé	Non
Est un microARN	Oui
Faible complexité	Non
Famille	f8
Liste des gènes cibles (accession, nom, position, score)	CJ871514,Human ribosomal DNA complete repeating unit,441,0.0
Logiciel de prédiction	HHMMiR|miPred
Longueur du petit ARN	21
Longueur du précurseur	88|102
Logiciel d'alignement	MAQ

Mésappariement ajouté	Oui
Mésappariements	1
MFE du précurseur	-22.9|-23.3
microARN réél ou pseudo selon *MiPred*	Pseudo
microARN*	GUGGUAGAAUUGCUCA
miRdup	Oui
Niveaux d'expression corrigés	237,283,139,242,115,212,167,118,72,113
Niveaux d'expression originaux	221,257,131,228,103,203,160,112,69,104
Nom	1557_1102_758
Nom du ribosome	Non ribosomal
Nombre de gènes cibles	1
Nombre de précurseurs uniques	2
Non codant	Non
Notes	N/A
Numéro d'accession de l'EST	gi|9360122|gb|BE400654.1|BE400654
Ontologie, composant cellulaire	Pas de référence UniProt dans le gène cible
Ontologie, fonction moléculaire	Pas de référence UniProt dans le gène cible
Ontologie, ID de la protéine	Pas de référence UniProt dans le gène cible
Ontologie, Nom de la protéine	Pas de référence UniProt dans le gène cible
Ontologie, processus biologique	Pas de référence UniProt dans le gène cible
Position du petit ARN	119
Qualité	0
Ribosomal	Non
Score de confidence de *MiPred*	73.4
Score de prédiction	0,939046239979188|0.094
Séquence couleur sans adaptateur	32022313123021010103 TGAGGTCTCGTAGCACACAT (format MAQ)
Séquence du petit ARN après alignement (microARN)	UAGGAGCAUGAUUCAACCAAU

Séquence du(des) précurseur(s)	AUUAGCAAGUUUGAGCUCGGUGGUAGAAUUGCUCAAUGAGAUCGUGAAUAGGAGCAUGAUUCAACCAAUUGACAUCAAUGUUGAUAAA| AUGAAAAACAAGGGAUUAGCAAGUUUGAGCUCGGUGGUAGAAUUGCUCAAUGAGAUCGUGAAUAGGAGCAUGAUUCAACCAAUUGACAUCAAUGUUGAUAAA
Structure du(des) précurseur(s)	(((((((...((((..(((((((((.(((((((((.(((....))).....)))))..)))))..)))).)))..)))))..)))))))))...| .....(((((((...((((..(((((((((.(((((((((.(((....))).....)))))..)))))..)))).)))..)))))..)))))))))...
Total des niveaux d'expression	1588
Total des niveaux d'expression corrigé	1698
Uniref	Aucun

CONCLUSIONS ET PERSPECTIVES

Dans cette étude nous avons développé une nouvelle approche pour identifier les microARNs chez le blé à partir de 10 librairies de petits ARNs séquencées par la plateforme AB SOLID. Ces petits ARNs ont été obtenus à partir de plusieurs variétés du blé cultivées sous des conditions normales et de stress abiotiques. Les fragments séquencés ont été alignés sur les ESTs du blé et leurs niveaux d'expression calculés dans chacune des librairies. Les précurseurs ainsi extraits sur les ESTs ont été repliés et soumis à deux logiciels de prédiction de précurseurs de microARNs, dont seuls les prédictions communes ont été retenues. Les gènes ciblés par les microARNs portés par les précurseurs prédits ont été déterminés et une série de filtres (non codants, faible complexité) a été appliquée, permettant d'éliminer des faux positifs. A cette étape, nous avons identifié, classé et caractérisé 3862 nouveaux microARNs potentiels. Les ontologies des gènes ciblés et l'expression différentielle entre les différentes conditions expérimentales ont aussi été identifiées. Enfin, une dernière étape a permit de valider la position de 1016 microARNs séquencés sur leur précurseur prédit, grâce à un outil d'apprentissage machine développé dans cette étude, *miRdup*. Cette approche à donc révélé qu'une grande quantité de microARNs est située dans les ESTs du blé, prédits avec une bonne confidence grâce à notre pipeline. Parmi les résultats, les profils d'expression déterminent directement dans la ou lesquelles conditions ils sont impliqués, que ce soit pendant l'exposition au froid, la salinité ou l'aluminium. De plus, dépendamment du tissu et du moment où ils ont été prélevés, certains microARNs n'ont pas la même abondance dans les racines et feuilles, ou en phase végétative et de floraison, indiquant une activité spécifique aux tissus. Au total, environ 200 microARNs différentiellement exprimés ont été identifiés ainsi que leurs gènes cibles, permettant alors de leur attribuer une fonction potentielle, jusqu'alors inconnue, notamment grâce à l'identification des ontologies de ces gènes. Il s'agit d'une grande avancée dans la recherche sur la régulation des gènes chez le blé ainsi que l'annotation des parties non codantes qui possèdent un microARN et son précurseur.

Plusieurs solutions ont été apportées dans cette étude pour répondre aux défis posés par la prédiction de microARNs à partir d'un séquençage haut débit. Du côté technique, la quantité de données produites par une telle étude a dû être gérée par des objets sérialisables, nous permettant ainsi de ne pas saturer la mémoire des ordinateurs à notre disposition. D'autre part, la prédiction croisée, comme utilisée dans ce manuscrit est l'une des façons de pallier en partie aux faiblesses des outils utilisés. De plus, plusieurs problèmes associés au génome du blé sont apparues. En effet, il n'est pas séquencé au complet, nous avons donc aligné nos séquences et recherché les gènes cibles dans les banques d'ESTs du blé. Ces banques sont dispersées dans plusieurs bases de données et ont donc du être rassemblées et homogénéisées. Par ailleurs, nous avons ajouté une étape de plus par rapport à la littérature pour prendre en compte la position des ARN séquencés, considérant que seul la prédiction du précurseur ne suffisait pas. Dans cette dernière étape, nous avons développé *miRdup*, basé sur du boosting avec Adaboost sur un arbre de décision C4.5, qui nous a permis d'ajouter un meilleur score à nos validations. A l'avenir, cet outil sera très utile pour augmenter la précision des prédictions de microARNs à grande échelle obtenus par séquençage à haut débit.

Parmi les perspectives, une meilleure compréhension des réseaux d'interaction entre les éléments régulateurs et leurs cibles permettra d'identifier des pistes pour adapter le blé à des régions du monde inaptes à sa culture (sols riches en aluminium, milieux froids ou salés). Par ailleurs, les microARNs étant très conservés dans l'évolution, ils pourront servir pour des recherches dans les espèces proches du blé. Du coté des outils bioinformatiques employés aux différentes étapes nécessitent certaines améliorations car nous avons constaté, tout comme d'autres études (Huang et al. 2007, Teune & Steger 2010, Williamson et al. 2012), qu'ils n'aboutissent pas aux mêmes résultats. De plus, de nombreux outils existent et remplissent les mêmes objectifs. Le manque de consensus sur la manière de prédire des structures secondaires, des microARNs, ou des cibles de microARNs rend le choix des logiciels complexe. Ce champ d'étude mérite donc encore des améliorations afin de standardiser les méthodes ou définir les meilleurs cas d'utilisation selon le matériel choisi. D'autre part, les structures secondaires des précurseurs publiés dans miRbase ne ressemblent pas toutes à celles obtenues avec *RNAfold*, notamment dû aux algorithmes de repliement différents (Parisien & Major 2008). Le repliement est donc une étape cruciale, et une

validation avec un autre programme de repliement pourrait être envisagée dans le futur. Il est important aussi d'indiquer que quelques précurseurs de miRbase ont montré des structures avec des épingles à deux boucles validées biologiquement et qui ne sont pas encore considérés par les algorithmes de prédiction. Ce dernier point devrait être implémenté dans les futurs programmes de prédiction de microARNs. Une telle étude pourrait être effectuée de nouveau sur un génome complet dans plusieurs années afin de détecter plus de microARNs. Toutefois, une autre approche serait d'aligner les fragments contre des génomes d'autres espèces proches du blé, comme *Brachypodium*, en accordant plus de mésappariements lors de l'alignement. Par ailleurs, nos données ou petits ARNs séquencés comporte d'autres types d'ARN régulateurs dont la fonction est inconnue tels que les très petites séquences de 16pb. Ce dernier point pourrait être une recherche future afin de caractériser une nouvelle classe de très petits ARNs. Finalement, pour confirmer les prédictions et donner du poids à notre pipeline, il est nécessaire d'en valider expérimentalement à l'aide de sondes et de la technique Norhern Blot (Chen et al. 2007). Concernant miRdup, ce classificateur a été entrainé sur miRbase, afin d'être plus souple quant à la découverte de nouveaux microARNs, mais il a aussi la possibilité d'être entrainé sur certaines espèces, telles que les plantes, les animaux, ou à des groupes spécifiques d'espèces. Ceci permettra de spécialiser l'impact d'un modèle dépendamment de la provenance des données.

Enfin, nous présentons en annexe une voie possible pour la mise en place du pipeline avec la plateforme Armadillo (Lord et al. 2012). Ce logiciel permet l'automatisation de tâches, et nous avons écris les différents scripts de cette étude dans le but de les y intégrer. Nous rendons donc reproductible cette étude et mettons à la disposition de la communauté, une plateforme complète de prédiction. Les capacités actuelles d'*Armadillo* sont tout à fait propices pour des études à large échelle, mais un grand travail est nécessaire pour satisfaire l'accès automatisé à des plateformes de calcul, ce qui n'est pas encore le cas. Une fois l'intégration d'un système de répartition de tâches implantée dans *Armadillo* pour optimiser ses capacités, la recherche de nouveaux microARNs à partir de NGS contenant des millions de fragments en profiterait grandement.

GLOSSAIRE

ADN : Macromolécule de poids moléculaire élevé, formée de polymères de nucléotides dont le sucre est le 2-désoxyribose. Il se présente sous forme d'une double chaîne hélicoïdale dont les deux brins sont complémentaires. Il constitue le génome de la plupart des organismes vivants.

ADN complémentaire : Ecrit aussi ADNc, simple brin artificiellement synthétisé à partir d'un ARNm, représente ainsi la partie codante de la région du génome qui a été transcrit en cet ARNm. Il est obtenu après une réaction de transcription inverse d'un ARNm mature.

Alignement de séquences : Manière de disposer les composantes des ADN, ARN ou acides aminés pour identifier les zones de concordances qui traduisent des similarités ou dissemblances. Des gaps peuvent être inclus dans l'alignement pour aligner les caractères communs sur des colonnes successives.

Anti-angiogenèse : Empêche l'angiogenèse qui est le processus actif complexe de formation de nouveaux capillaires sanguins à partir de vaisseaux sanguins existants

Anti-apoptotique : Empêche l'apoptose qui représente la mort cellulaire programmée des cellules métaboliquement actives et qui est contrôlée par des mécanismes d'induction de la cellule.

Apoptose : représente la mort cellulaire génétiquement régulée qui a lieu dans des cellules métaboliquement actives et qui est contrôlée par des mécanismes d'induction.

ARN : Macromolécule formée par la polymérisation de nombreux nucléotides dont le sucre est le ribose, présente dans le cytoplasme, les mitochondries ainsi que dans le noyau cellulaire, et servant d'intermédiaire dans la synthèse des protéines.

ARN messager mature : Correspond aux exons de l'ARNm mis bout à bout après l'épissage.

ARN messager (ARNm) : Copie transitoire d'une portion de l'ADN correspondant à un ou plusieurs gènes utilisé comme intermédiaire par les cellules pour la synthèse des protéines.

Biopuce ADN : Ensemble de molécules d'ADN fixées en rangées ordonnées sur une petite surface permettant d'analyser le niveau d'expression de gènes (transcrits) d'un échantillon.

BLAST : Méthode de recherche heuristique utilisée en bio-informatique permettant de rechercher les régions similaires entre deux ou plusieurs séquences de nucléotides ou d'acides aminés.

Bootstrap : Technique d'inférence statistique basée sur une succession de rééchantillonnages.

Codon : Triplet de nucléotides A, C, U ou G de l'ARN messager.

Complexe RISC : *RNA-induced silencing complex.* Complexe protéique impliqué dans le mécanisme des microARNs.

Contig : Séquence consensus provenant de l'assemblage d'ADN clonés contigus souvent obtenus par séquençage.

Cotylédons : les feuilles primordiales constitutives de la graine.

Cytoplasme : Désigne le contenu d'une cellule vivante. Plus exactement, il s'agit de la totalité du matériel cellulaire du protoplasme délimité par la membrane plasmique.

Dicer : Endoribonuclease de la famille RNase III qui clive les ARN double brins et les précurseurs en petites fractions double brin appelés ARNs interférents d'environ 20 à 25pb de long.

Dicotylédones : Plante à fleur possédant deux cotylédons.

Électrophorèse : Technique utilisée en biologie pour la séparation et la caractérisation des molécules.

emPCR : PCR en émulsion.

Épissage : Chez les eucaryotes, processus par lequel les ARN transcrits à partir de l'ADN génomique peuvent subir des étapes de coupure et ligature qui conduisent à la suppression de certaines régions dans l'ARNm. Les segments conservés s'appellent des exons et ceux qui sont éliminés s'appellent des introns.

EST : *Expressed sequence tag*, ou marqueur de séquence exprimée. Courte portion séquencée d'un ADN complémentaire, utilisée comme marqueur pour différencier les gènes entre eux dans une séquence ADN et identifier les gènes homologues dans d'autres espèces.

Eucaryote : Organismes dont les cellules possèdent un noyau.

Exon : Parties transcrites des gènes qui codent des protéines.

Fragment : *read* en anglais, courte séquence d'environ 35 à 50pb à l'état brut après son séquençage, en séquence couleur au lieu de nucléotides pour ABI SOLID.

Gap : Un gap apparait dans un alignement entre deux séquences pour représenter une délétion dans l'une d'elle ou une insertion dans l'autre au niveau d'une paire de bases donnée.

Génome : ensemble du matériel génétique d'un individu ou d'une espèce codé dans son ADN (à l'exception de certains virus dont le génome est porté par des molécules d'ARN).

Hématopoïèse : Ensemble des processus par lesquels les cellules souches pluripotentes de la moelle osseuse, soit les cellules souches lymphoïdes et les cellules souches myéloïdes se multiplient, se différencient et aboutissent à des cellules sanguines matures.

Hérédité : Transmission de caractéristiques d'une génération à la suivante, quel que soit le mode de cette transmission.

Histone : Protéine autour de laquelle l'ADN s'enroule pour former les chromosomes.

Indexage : Permet de retrouver une information plus rapidement et facilement

Intergénique : Séquence d'ADN non transcrit séparant les gènes.

Intron : Portion d'un gène non codante éliminée lors de l'épissage.

Intronique : Se dit d'une région contenant un intron d'une séquence avant son épissage.

Maturation de l'ARNm : Étapes permettant le passage de l'ARN à l'ARNm vers l'établissement d'une protéine.

Métastase : Foyer secondaire d'une affection, suppuration et surtout cancer, disséminée par voie sanguine ou lymphatique à partir d'un foyer primitif

Mimicry : Utilisé dans la recherche des gènes cibles de microARNs, consiste en un ARN non clivable qui forme une interaction spécifique sous la forme d'une hernie avec une séquence complémentaire.

Mitose : Désigne les évènements chromosomiques de la division cellulaire.

Monocotylédone (ou Monocotylédon, ou Monocots) : Parmi les angiospermes ou plantes à fleurs, comprennent des végétaux dont la plantule typique ne présente qu'un seul cotylédon sur l'embryon, qui évolue en donnant une préfeuille.

Nucléoside : Glycosylamines constitués d'une nucléobase (base) liée à un ribose ou un désoxyribose via une liaison glycosidique. Parmi les nucléosides, on retrouve par exemple la cytidine, l'uridine, l'adénosine, la guanosine, la thymidine ou encore l'inosine. Dans les cellules, les nucléosides peuvent être phosphorylés par des kinases spécifiques, permettant la formation des nucléotides qui sont les éléments constitutifs de l'ADN et de l'ARN.

Nucléotide : Molécule organique composée d'une nucléobase, d'un pentose et de 1 à 3 groupements phosphates. Certains nucléotides forment la base de l'ADN et de l'ARN.

Nucléotides fluorescents : Nucléotides formant des séquences d'ADN complémentaires de la séquence d'un gène ou d'une partie du génome préparées in vitro et couplées à des fluorochromes (Colorisation pour reconnaissance visuelle).

Oligo : Un petit polymère de deux à vingt nucléotides.

Ovocyte : Cellule sexuelle femelle des métazoaires. Seuls quelques-uns évolueront en ovules après maturation.

Pathologie : science qui a pour objet l'étude des maladies et notamment leurs causes et leurs mécanismes.

PCR (*Polymerase Chain Reaction*) : Méthode de biologie moléculaire d'amplification génique in vitro, qui permet de copier en grand nombre une séquence d'ADN ou d'ARN connue à partir d'une faible quantité d'acide nucléique.

Pipeline : Méthode ou modèle d'exécution séquentielle de tâches dans lequel le résultat d'une tâche consiste en l'entrée de la ou des tâches subséquentes.

Plasticité : En biologie de l'évolution, un caractère est dit plastique s'il varie en fonction de l'environnement où l'individu se trouve ou au cours de l'ontogénie de cet individu. La plasticité peut s'observer, le développement, l'anatomie, la morphologie, le comportement, etc. On peut ainsi dire que plus un phénotype est plastique, moins il est déterminé (et contraint) génétiquement.

Pluripotence : faculté de certaines cellules à se différencier en tout type cellulaire d'un organisme

Polymérase : Enzymes qui ont pour rôle la synthèse d'un brin de polynucléotide (ADN ou ARN), le plus souvent en utilisant un brin complémentaire comme matrice et des nucléotides triphosphosphate (NTP ou dNTP) comme monomères

Polymorphisme : Coexistence dans une population de plusieurs types génétiques différant les uns des autres par des caractères observables divers.

Pré-miRNA : appelé aussi précurseur, est la séquence en forme d'épingle qui porte le microARN.

Primer : Courte séquence d'ARN ou d'ADN, complémentaire du début d'une matrice, servant de point de départ à la synthèse du brin complémentaire de cette dernière par une ADN polymérase.

Protiste : Désigne les eucaryotes autres que les animaux (Métazoaires), champignons (Eumycètes), et plantes (des Embryophytes aux Archaeplastida, selon les définitions).

Régulation post-transcriptionnelle : Phase de la régulation de l'expression des gènes comprenant tous les mécanismes affectant directement les molécules d'ARN produite lors de la transcritpion.

Réplication de l'ADN : processus au cours duquel l'ADN est synthétisé grâce à l'ADN polymérase. Ce mécanisme permet à l'ADN d'être dupliqué (donc doublé).

Ribosome : Complexes ribonucléoprotéiques (c'est-à-dire composés de protéines et d'ARN) présents dans les cellules eucaryotes et procaryotes. Leur fonction est de synthétiser les protéines en décodant l'information contenue dans l'ARN messager.

Scission : Reproduction par scissiparité ou division binaire, qui est un mode de multiplication asexué qui se réalise simplement par division de l'organisme.

Séquence : Suite des acides aminés d'un peptide (structure primaire) ou suite des nucléotides d'un ADN ou d'un ARN, ou d'un segment de ceux-ci. Par convention, une structure de nucléotides est écrite de l'extrémité 5' à 3'

SNP (*Single-Nucleotide Polymorphism*) : Désigne le polymorphisme d'un seul nucléotide, qui est la variation (polymorphisme) d'une seule paire de bases du génome, entre individus d'une même espèce.

Synthèse protéique : Acte par lequel une cellule assemble une chaîne protéique en combinant des acides aminés isolés présents dans son cytoplasme, guidé par l'information contenue dans l'ADN. Elle se déroule en deux étapes au moins : la transcription de l'ADN en ARN messager et la traduction de l'ARN messager en une protéine.

Traduction : Interprétation des codons de l'ARNm en acides aminés.

Transcriptome : Ensemble des ARN issus de l'expression d'une partie du génome d'un tissu cellulaire ou d'un type de cellule.

Transformation Burrows-Wheeler : Technique utilisée en compression de données inventée par Michael Burrows et David Wheeler (Burrows & Wheeler 1994).

Vernalisation : En botanique, c'est l'acquisition de la capacité des plantes à fleurir ou germer au printemps par l'exposition prolongée au froid de l'hiver.

ANNEXES

ANNEXE A : PLATEFORME D'AUTOMATISATION DE TÂCHES ARMADILLO

Dès le commencement de ce projet, le volume des données à analyser, transformer et organiser nous est apparu considérable. Par exemple, nous avions en entrée du pipeline d'analyse, 19 giga-octets de données brutes de séquençage (fragments en séquence couleur) et environ 1 giga-octets pour la base de données des ESTs provenant du regroupement de plusieurs bases de données bioinformatiques locales et distantes (voir section 1.2). Cette quantité de données n'est en soit pas très importante, cependant, pour profiter des 48 ordinateurs en grappes disponibles pour cette étude (12 x 2 cpu + 4 giga-octets de mémoire vive (RAM) par ordinateur, 16 x 8 cpu + 8 giga-octets RAM, 20 x 4 cpu + 8 giga-octets RAM), la copie locale de tous ces fichiers sur chacun des ordinateurs était irréalisable par leur architecture. De plus, la quantité de données générées par notre pipeline d'analyse jusqu'à maintenant s'élève à 205 giga-octets. Dans cette optique, plusieurs alternatives ont été considérées pour la mise en place d'un pipeline d'analyse des miRNAs, permettant à la fois un déploiement rapide mais aussi la réplication et l'analyse de nouveaux résultats au fur et à mesure que les données de séquençage allaient nous parvenir. J'ai ainsi participé au développement d'un système de gestion des flux de travail : *Armadillo* version 1.0[1] (publication soumise à la revue Plos One, en annexe), permettant la mise en place de protocoles d'analyses bioinformatiques standardisés, reproductibles et facilement transposables d'une architecture locale (ordinateur personnel) à une architecture distribuée (grappe d'ordinateurs). Nous décrivons dans cette section ma contribution à l'élaboration de ce système de gestion de flux de travail (SGFT). De plus, l'élaboration en parallèle d'un pipeline d'analyse permettant la prédiction des miRNAs à partir de données de séquençages de type SOLID sous la forme d'un flux travail, ou *workflow*, a été réalisé (Figure A1), et fera l'objet d'une publication (en cours de réalisation).

---

[1] Disponible à l'adresse : http://adn.bioinfo.uqam.ca/armadillo

a. Les flux de travail

Les flux de travail sont une représentation de l'ordonnancement des tâches permettant de définir formellement (Aalst & Hee 2004). La gestion de ces flux de travail s'effectue à l'aide de plateformes de gestion de flux de travail. Ces systèmes permettent la création, la mise en œuvre et la coordination des processus et des applications inclus dans les flux de travail (Beaulah et al. 2008). Les objectifs spécifiques généralement admis des SGFT est d'automatiser des tâches pouvant produire des erreurs, intégrer l'analyse et la visualisation des données générées, permettre la collection, faciliter le l'organisation et la passage à réorganisation des échelle des (reformatage), simplifier la déploiement et le une plus grande procédés et enfin

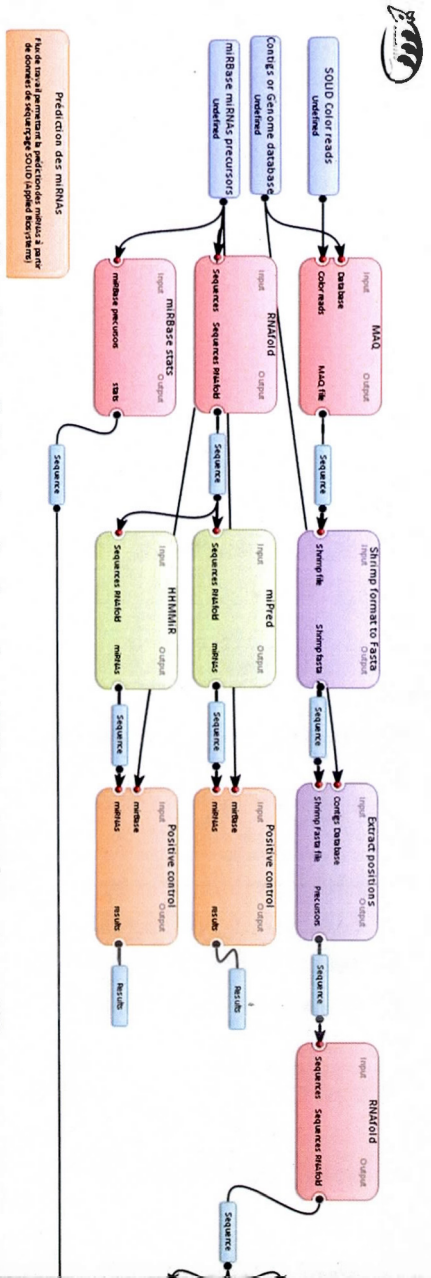


Figure A1 : Flux de travail de recherche des microARNs implanté sur Armadillo

compréhension et la cognition du procédé (Woollard et al. 2008).

L'avantage de l'utilisation d'une telle plateforme dans la conduite d'analyses scientifiques est que le chercheur n'a pas besoin d'apprendre un langage de programmation pour modifier l'analyse (Goderis & University of Manchester. School of computer 2008). Ainsi, les étudiants ou les scientifiques peuvent développer eux- mêmes, avec un minimum d'effort, des prototypes de flux de travail permettant l'automatisation de certaines tâches sur leur poste de travail, sans avoir obligatoirement besoin d'une connexion réseau. De plus, dans le cadre de notre projet, ceci a permis de permettre la contribution de plusieurs acteurs dans la création du pipeline d'analyse.

b. Les systèmes de flux de travail

Avant la conception de ce système de flux de travail, plusieurs alternatives déjà présentes dans ce domaine de la bioinformatiques ont été évaluées (Dinov et al. 2011) tel que Galaxy (Giardine et al. 2005), Kepler (Bowers et al. 2008), LONI (Dinov et al. 2011) , Taverna (Oinn et al. 2007), et Triana (Taylor et al. 2007). La plupart de ces SGFT sont très similaires dans leur utilisation. Ils visent principalement une plus grande utilisation du pouvoir computationnel, l'intégration des services en lignes (*web*) et un accès de plus en plus important au « nuage » (*cloud computing*). Cependant, aucun de ces systèmes ne convenait à notre domaine d'application. D'une part, Galaxy, une plateforme en ligne orientée vers l'étude des données de séquençage à haut débit (Schatz 2010) ne supportait par nativement les outils nécessaires à notre analyse. De plus, envoyer notre volume de données en ligne était impensable et bien que pouvant être installé sur un ordinateur personnel, Galaxy ne permet pas directement la distribution de tâches sur une grappe d'ordinateurs. Ce n'est que tout récemment qu'une version permettant l'utilisation du « nuage » de *Amazon* (*Amazon Cloud EC2*) a été mise à la disposition de la communauté scientifique (Afgan et al. 2010). D'autre part, la plateforme Taverna, orientée vers l'utilisation des services en lignes pour l'exécution des logiciels excluait les grappes d'ordinateurs mises à notre disposition pour la réalisation de cette étude, ne permettant pas les connections avec les ports normalement utilisés pour ce genre de services. Encore, Kepler, Triana et LONI, supportant l'exécution de programmes Java localement, ont été proposés pour notre étude. Cependant, sans

documentation disponible et LONI ne permettant pas l'ajout d'applications externes et requérait la création d'un serveur d'applications, il a plutôt été proposé de développer notre plateforme d'analyse, utilisant le langage multiplateforme Java, permettant de répondre spécifiquement à nos spécifications désirées.

Ces caractéristiques désirées de système de gestion de flux de travail permettant de réaliser notre pipeline d'analyse étaient principalement :

1. Communication avec des bases de données distantes (ex. miRBase).
2. Gestion de l'exécution des logiciels et des erreurs permettant la reprise de certaines analyses au besoin.
3. La possibilité d'ajouter ou de retirer rapidement des logiciels au besoin e.g. si une analyse particulière s'avérait non concluante tel que dans le cas de *SHRiMP*.
4. Conservation de la trace des analyses pour vérification subséquente.

c. Armadillo

Tel que d'écrit dans une étude antérieure (Stevens et al. 2003), le cycle de vie des expériences *in silico* consistent en plusieurs points qui doivent être adressés tels que : le design initial de l'expérience, le schéma d'exécution, l'interprétation des résultats obtenus et la présentation et publication de ces résultats. Ainsi, quand ce projet a été entamé sous la direction d'Etienne Lord (étudiant au doctorat), nous avons décidé d'intégrer la plupart de ces éléments dans la conception de la plateforme *Armadillo*. Pour cette première phase de développement (Figure), j'ai participé à la conception du noyau de *Armadillo* répondant à ces critères de conception (ci-haut), à la réalisation des classes qui permettent à Java d'accéder au système de gestion de base de données (SGBD) local SQLite (Hipp 2003), servant de support à l'application, à la création de l'interface du site web du SGFT ainsi qu'à la génération automatique de rapports de tâches en langage HTML. Finalement, j'ai été responsable du développement du code Java permettant la lecture d'arbres phylogénétiques.

Lors du développement de la plateforme, le système d'exploitation *Windows* a été choisi au départ pour permettre une plus grande utilisation dans les différents laboratoires, cependant, l'application étant développée dans le langage Java, son utilisation dans ce projet s'est

déroulée sous Linux. Il ne nous a fallu que d'ajouter les exécutables adéquats dans notre distribution. Les applications les plus utilisées en bioinformatique et les types de données les plus courant e.g. Fasta et Phylip ont été incorporés par défaut dans la plateforme. De cette manière les chercheurs peuvent ignorer comment se déroule l'exécution et plutôt mettre l'emphase sur le pourquoi de l'exécution de cette chaîne de programme i.e. l'hypothèse de base qu'ils veulent prouver (Stevens et al. 2007). La plateforme a ainsi été conçue autour d'un concept « *What you See is What you Pipe* » (WYSIWYP). Comme LONI, Galaxy et Taverna, les composants du flux de travail représentant les données ou les applications peuvent être reliés entre eux en utilisant le principe du « *drag-and-drop* » à partir d'une boite d'outils (Figure A2A) vers l'éditeur de flux de données (Figure A2B). La plateforme inclus aussi différentes méthodes de visualisation des données utilisées ou générées telles qu'une visualisation de séquences statiques (Figure D) ainsi que des logiciels permettant de manipuler et d'inférer des arbres phylogénétiques (Figure A2C). La configuration des applications est quant à elle réalisée à l'aide de boîtes de dialogue (Figure A2E) qui ont été conçues de manière à permettre l'accès aux options les plus communes.

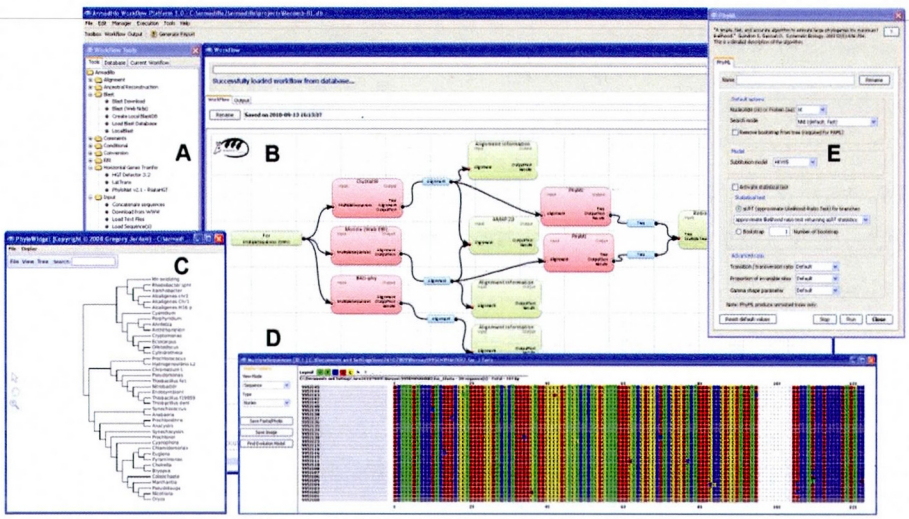


Figure A2 : Vue de l'interface utilisateur de Armadillo version 1.0

L'orientation de développement de la plateforme *Armadillo* est présentement de permettre des études phylogénomiques i.e. l'analyse de l'évolution des espèces en tenant compte de la totalité de l'information génomique. Ainsi, on pourra à terme l'utiliser pour la ré-analyse de certains des résultats de la présente étude en prenant compte de l'évolution de miRNAs, comme la recherche de transfert horizontaux des miRNAs entre les différentes espèces proches de *Triticum aestivum* et non plus seulement l'analyse de leurs présence, absence et niveaux d'expression dans diverses conditions expérimentales.

d. Pipeline d'analyse de la prédiction des microARNs

Dans la deuxième phase de ce projet, le code Java a été adapté pour permettre son exécution sur la plateforme *Armadillo.* À terme, avec la quantité de données générées durant ce type d'analyse, ce flux de travail pourra être couplé directement à une interface web et à un SGBD plus important e.g. MySQL (www.mysql.com) ou Oracle (Entreprise Edition 11g, www.oracle.com). Ce faisant, des projets plus ambitieux pourraient être entrepris. Encore, puisque de nouvelles données sont quotidiennement ajoutées aux banques de données de microRNA et aux banques de séquences (e.g. NCBI), une réexécution du flux de travail développé devrait être réalisé annuellement afin de rechercher la présence d'autres microARNs conservés.

En conclusion, la dernière partie du pipeline d'analyse est en court de réalisation sous forme de flux de travail. Le dernier avantage de cette approche par flux de travail est que la majeure partie du pipeline originel développé en Java est assez générique dans sa conception, mais requiert des changements significatifs pour être réutilisé avec d'autres jeux de données. Cependant, en ayant pu diviser le code Java en différents éléments fonctionnels (objets dans le flux de travail), il est dorénavant envisageable de se servir de ceux-ci dans d'autres types d'études. On peut par exemple penser à utiliser l'objet permettant la recherche des Ontologies (basé sur le programme Java ii_geneOntology.java) dans plusieurs études en bioinformatique. Enfin, puisque notre flux de travail est générique, un objet pourrait seulement être ajouté dans le flux de travail, ce qui permettrait la lecture de d'autres sources de données de séquençage comme les données provenant du séquenceur Illumina, pour l'adapter à d'autres études.

ANNEXE B : CODE SOURCE ET RESULTATS BRUTS DU PROGRAMME MIRDUP

Code source

Le code source est composé de 2 classes : formatNewFiletoModel.java et features.java. La classe formatNewFiletoModel.java permet de rendre compatible les fichiers d'entrée pour Weka. Cette classe fait appel à l'autre classe, features.java, qui permet d'extraire tous les attributs qui caractérisent le duplexe du microARN et sa région complémentaire.

formatNewFiletoModel.java

```
/*
 * Prepare le fichier pour weak. Prend un fichier fasta en entrée avec les microARNs de miRbase,
et un autre avec les precurseurs replies et leur sequence.
 */
package mirnas;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Random;
//import weka.filters.*;

/**
 *
 * @author Mickael
 */
public class formatNewFiletoModel {

     /**
     * @param args the command line arguments
     */
    public static void formatNewFiletoModel(String[] args) {
        //formatfastaFiles();
        createDatasets();


    }

    private static void formatfastaFiles() {
        File mirnaFile = new File("monocots.miRbase.mirnas.fasta");
        File precFile = new File("monocots.miRbase.prec.fold.fasta");
        File outfile = new File("monocots.miRbase.mirnaPrecs.txt");

        HashMap<String,String> hmMirnas = new HashMap<String, String>();
        HashMap<String,String> hmPrec = new HashMap<String, String>();

        try {
            BufferedReader br = new BufferedReader(new FileReader(mirnaFile));
            String line="";
            while (br.ready()){
                line=br.readLine();
                if (line.startsWith(">")){
                    hmMirnas.put(line.substring(1, line.indexOf(" ")).toLowerCase(),
br.readLine());
                }
            }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }

        //prec and fold
        try {
            BufferedReader br = new BufferedReader(new FileReader(precFile));
            String line="";
            while (br.ready()){
                line=br.readLine();
                if (line.startsWith(">")){
                    String m = line.substring(1, line.indexOf(" ")).toLowerCase();
                    String p = br.readLine();
                    String s = br.readLine();
                    String struct = s.substring(0,s.indexOf(" "));
                    hmPrec.put(m,p+"\t"+struct);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            PrintWriter pw = new PrintWriter(new FileWriter(outfile));
            for (String name : hmMirnas.keySet()) {
                String mirna = hmMirnas.get(name);
                name=name.replace("-5p", "").replace(".1", "").replace(".2", "").replace(".3",
"");
                String prec = hmPrec.get(name);
                if (prec!=null) {
                    pw.println(mirna + "\t" + prec);
                }
            }
            pw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
    /**
     * @param args the command line arguments
     */
    public static void createDatasets() {
        //features.executeWindowsCommand("PATH=$PATH:/Users/mickael/Downloads/ViennaRNA-
1.8.5/Progs/");
        File truepremirnasTrain = new File("monocots.miRbase.mirnaPrecs.txt");
        File falsepremirnasTrain=new File("monocots.miRbase.mirnaPrecs.txt");
        File outfileWekaTrain=new File("wekaFileTrainingSetmonocots.BestFeatures.arff");

//        File truepremirnasTrain = new File("positive.premirnas.TrainingSet.txt");
//        File falsepremirnasTrain=new File("positive.premirnas.TrainingSet.txt");
//        File outfileWekaTrain=new File("wekaFileTrainingSet.BestFeatures.arff");
//
//
//        File truepremirnasTest = new File("positive.premirnas.TestSet.pmrd.txt");
//        File falsepremirnasTest=new File("positive.premirnas.TestSet.pmrd.txt");
//        File outfileWekaTest=new File("wekaFileTestSet.BestFeatures.pmrd.arff");
//
//        File predictedpremirnas=new File("predicted.premirnas.txt");
//        File outfileWekaPred=new File("wekaFilePredSet.BestFeatures.arff");

        try {
             PrintWriter pw;
//            // Training set
            pw = new PrintWriter(new FileWriter(outfileWekaTrain));
            pw.println(getBestWekaFileEntries());
            System.out.println("Get training positive dataset features");
            for (String s : generateData(truepremirnasTrain,true)) {
                pw.println(s);
            }
            pw.flush();
            System.out.println("\nGet training negative dataset features");
            for (String s : generateData(falsepremirnasTrain,false)) {
                pw.println(s);
            }
            pw.println();
```

```
            pw.close();

              // Test set
//            pw = new PrintWriter(new FileWriter(outfileWekaTest));
//            pw.println(getBestWekaFileEntries());
//            System.out.println("Get test positive dataset features");
//            for (String s : generateData(truepremirnasTest,true)) {
//                pw.println(s);
//            }
//            pw.flush();
//            System.out.println("\nGet test negative dataset features");
//            for (String s : generateData(falsepremirnasTest,false)) {
//                pw.println(s);
//            }
//            pw.println();
//            pw.close();


            // Predicted set
//            pw = new PrintWriter(new FileWriter(outfileWekaPred));
//            pw.println(getBestWekaFileEntries());
//            System.out.println("\nGet predicted dataset features");
//            for (String s : generateData(predictedpremirnas,true)) {
//                pw.println(s);
//            }
//            pw.println();
//            pw.close();

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }


    /**
     *  Generate DataSet, positive or negative, from an infile
     * @param infile
     * @param positiveDataset
     * @return Arraylist of features for each iteration in Weka format (separated by ,)
     * positive dataset must be: mirna"\t"prec"\t"struc
     * negative dataset must be: prec"\t"struc  ; mirna will be randomly generated
     *
     */
    public static ArrayList<String> generateData(File infile, Boolean positiveDataset){
        ArrayList<String> al=new ArrayList<String>();
        int cpt=0;
            try {
                BufferedReader br = new BufferedReader(new FileReader(infile));
                String line="";
                while (br.ready()){
                    line=br.readLine();
                    String mirna="";
                    try {
                        String tab[] = line.split("\t");
                        mirna = tab[0];
                        if (mirna.contains("_")) {
                            mirna=mirna.split("_")[0];
                        }
                        String prec = tab[1];
                        String struc = tab[2];
                        features f;
                        if(positiveDataset){
                            f = new features(mirna, prec, struc, true);
                        } else{
                            mirna = mirnaNegativeData(mirna, prec, struc); //generate new mirna
                            f = new features(mirna, prec, struc, false);
                        }

                        if (!f.isMultipleLoop()) {
                            al.add(f.toStringBestAttributes());
                        }
                        cpt++;
                        if (cpt%1000==0) System.out.print("*");
                    } catch (Exception e) {
                        //System.err.println(mirna);
                    }
```

```
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }

    return al;
}


public static String mirnaNegativeData(String mirna, String precursor, String structure){
    Random r = new Random();
    int mirnalength=mirna.length();
    int start=r.nextInt(precursor.length()-mirnalength);
    String newmirna=precursor.substring(start,start+mirnalength);
    return newmirna;
}


public static void printFeatures(features f){
    p(f.getLength());
    p(f.getGCperc());
    p(f.getMaximumLengthWithoutBulges());
    p(f.getMaximumLengthWithoutBulgesPerc());
    p(f.getStartLengthWithoutBulges());
    p(f.getBasePairsInDuplexMirnaMirnaStar());
    p(f.getPresenceOfPerfect20MerBasePair());
    p(f.getStartOfPerfect20MerBasePair());
    p(f.getPresenceOfPerfect10MerBasePair());
    p(f.getStartOfPerfect10MerBasePair());
    p(f.getPresenceOfPerfect5MerBasePair());
    p(f.getStartOfPerfect5MerBasePair());
    p(f.getPresenceOfA());
    p(f.getPresenceOfU());
    p(f.getPresenceOfG());
    p(f.getPresenceOfC());
    p(f.getDistanceFromTerminalLoop());
    p(f.getDistanceFromHairpinStart());
    p(f.getMirnaIncludedInLoop());
    p(f.getLengthOfOverlapInLoop());
    p(f.getAverageNumberOfPairedBasesInWindow7());
    p(f.getAverageNumberOfPairedBasesInWindow5());
    p(f.getAverageNumberOfPairedBasesInWindow3());
    p(f.getBulgeAtPosition2());
    p(f.getBulgeAtPositionMinus2());
    p(f.getBulgeAtPosition1());
    p(f.getBulgeAtPositionMinus1());
    p(f.getNumberOfBulges());
    p(f.getLenghtOfBiggestBulge());
    p(f.getLengthBiggestBulgesPerc());

    System.out.println("");
}


public static void p(String s){
    System.out.println(s);
}
public static void p(int s){
    System.out.println(s);
}
public static void p(double s){
    //System.out.println(df.format(s));
}
public static void p(boolean b){
    System.out.println(b);
}

public static String getAllWekaFileEntries(){
    String s = ""
            + "@relation mirnaInPrecursor"+ "\n"
            + "@attribute length real"+ "\n"
            + "@attribute mfe real"+ "\n"
            + "@attribute GCperc real"+ "\n"
            + "@attribute MaximumLengthWithoutBulges real"+ "\n"
            + "@attribute MaximumLengthWithoutBulgesPerc real"+ "\n"
            + "@attribute StartLengthWithoutBulges real"+ "\n"
```

```
                + "@attribute BasePairsInDuplexMirnaMirnaStar real"+ "\n"
                + "@attribute PresenceOfPerfect20MerBasePair { true, false}"+ "\n"
                + "@attribute StartOfPerfect20MerBasePair real"+ "\n"
                + "@attribute PresenceOfPerfect10MerBasePair { true, false}"+ "\n"
                + "@attribute StartOfPerfect10MerBasePair real"+ "\n"
                + "@attribute PresenceOfPerfect5MerBasePair { true, false}"+ "\n"
                + "@attribute StartOfPerfect5MerBasePair real"+ "\n"
                + "@attribute PresenceOfA { true, false}"+ "\n"
                + "@attribute PresenceOfU { true, false}"+ "\n"
                + "@attribute PresenceOfG { true, false}"+ "\n"
                + "@attribute PresenceOfC { true, false}"+ "\n"
                + "@attribute PercOfA real"+ "\n"
                + "@attribute PercOfU real"+ "\n"
                + "@attribute PercOfG real"+ "\n"
                + "@attribute PercOfC real"+ "\n"
                + "@attribute DistanceFromTerminalLoop real"+ "\n"
                + "@attribute DistanceFromHairpinStart real"+ "\n"
                + "@attribute MirnaIncludedInLoop { true, false}"+ "\n"
                + "@attribute LengthOfOverlapInLoop real"+ "\n"
                + "@attribute AverageNumberOfPairedBasesInWindow7 real"+ "\n"
                + "@attribute AverageNumberOfPairedBasesInWindow5 real"+ "\n"
                + "@attribute AverageNumberOfPairedBasesInWindow3 real"+ "\n"
                + "@attribute BulgeAtPosition2 { true, false}"+ "\n"
                + "@attribute BulgeAtPositionMinus2 { true, false}"+ "\n"
                + "@attribute BulgeAtPosition1 { true, false}"+ "\n"
                + "@attribute BulgeAtPositionMinus1 { true, false}"+ "\n"
                + "@attribute NumberOfBulges real"+"\n"
                + "@attribute LenghtOfBiggestBulge real"+ "\n"
                + "@attribute LengthBiggestBulgesPerc real"+ "\n"
                + "@attribute Class { true, false}"+ "\n"
                + "@data";
        return s;
    }

    public static String getBestWekaFileEntries(){
        String s = ""
                + "@relation mirnaInPrecursor"+ "\n"
                + "@attribute DistanceFromTerminalLoop real"+ "\n"
                + "@attribute LengthOfOverlapInLoop real"+ "\n"
                + "@attribute AverageNumberOfPairedBasesInWindow3 real"+ "\n"
                + "@attribute BasePairsInDuplexMirnaMirnaStar real"+ "\n"
                + "@attribute LengthBiggestBulgesPerc real"+ "\n"
                + "@attribute AverageNumberOfPairedBasesInWindow5 real"+ "\n"
                + "@attribute LenghtOfBiggestBulge real"+ "\n"
                + "@attribute AverageNumberOfPairedBasesInWindow7 real"+ "\n"
                + "@attribute MirnaIncludedInLoop { true, false}"+ "\n"
                + "@attribute DistanceFromHairpinStart real"+ "\n"
                + "@attribute StartOfPerfect5MerBasePair real"+ "\n"
                + "@attribute MaximumLengthWithoutBulges real"+ "\n"
                + "@attribute MaximumLengthWithoutBulgesPerc real"+ "\n"
                + "@attribute mfe real"+ "\n"
                + "@attribute Class { true, false}"+ "\n"
                + "@data";
        return s;
    }

}
```

Features.java

```
/*
 * Recupere les atttributs a partir des microARNs, precurseurs et structures.
 * Attributs :
 * Longueur totale
 * MFE du duplexe
 * GC
 * Longueur maximale sans hernies
 * Paires de bases dans le duplexe miRNA-miRNA*
 * Depart d'un 20mer en bases pairées parfait
 * Depart d'un 8mer en bases pairées parfait suivit par 95% 12mer bases pairées
 * Presence de U
 * Distance a partir de la boucle terminale
 * Distance à partir du départ de l'epingle a cheveux
 * Longueur de la superposition sur la boucle
```

```
 * Nombre de paire de bases dans une fentre de 7nt
 * Nombre de paire de bases dans une fentre de 5nt
 * Nombre de paire de bases dans une fentre de 3nt
 * Hernie a la position 2
 * Hernie a la position -2
 * Hernie a la position 1
 * Hernie a la position -1
 * Nombre de hernies
 * Longueur maximale de la plus grande hernie
 */
package mirnas;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.text.DecimalFormat;
import java.util.ArrayList;

/**
 *
 * @author Mickael
 */
public final class features {

     String mirna;
     String prec;
     String precStruc;
     String mirnaStruc;
     String mirnastar;
     Boolean positive;
     public static DecimalFormat df = new DecimalFormat ( ) ;


    features(String miRNA, String precursor, String secondaryStructure, Boolean positiveDataset){
        mirna=miRNA;
        prec=precursor;
        precStruc=secondaryStructure;
        positive=positiveDataset;
        mirnaStruc=precStruc.substring(prec.indexOf(mirna), prec.indexOf(mirna)+mirna.length());
        //mirnastar=getMirnaStar();
        df.setMaximumFractionDigits(2);
    }

    //MFE
    public String getMFE(){
        String mfe="";
        try {
            mirnastar = getMirnaStar();
            if (mirnastar.equals("loop")||mirnastar.equals("error")){
                return "?";
            }
            File f = new File("foldtmp.txt");
            PrintWriter pw = new PrintWriter(new FileWriter(f));
            pw.write(mirna+"\n"+mirnastar);
            pw.close();
            String out = executeLinuxCommand("/ibrixfs1/Data/mik/tools/ViennaRNA-
1.8.4/Progs/RNAduplex < "+f.getAbsolutePath());
            mfe=out.substring(out.indexOf(" (")+2,out.length()-1);
            if (Double.valueOf(mfe)>0){
                mfe="?";
            }
            return mfe;
        } catch (Exception e) {
            return "?";
        }

    }

    public  int getLength(){
        return mirna.length();
    }
```

```
    //Pourcentage GC
    public  double getGCperc(){
        int l = mirna.length();
        int gc=0;
        for (int i = 0; i < l; i++) {
            if (mirna.charAt(i)=='C'||mirna.charAt(i)=='G'){
                gc++;
            }
        }
        double perc=(gc*100)/l;
        return perc;
    }

    //Longueur maximale sans hernies
    public  int getMaximumLengthWithoutBulges(){
        int l=0;
        int max=0;
        for (int i = 0; i < mirnaStruc.length(); i++) {
            if (mirnaStruc.charAt(i)=='('||mirnaStruc.charAt(i)==')'){
                l++;
            }else {
                l=0;
            }
            if (max<l) max=l;
        }
        return max;
    }

    //Longueur maximale sans hernies en pourcentage
    public  double getMaximumLengthWithoutBulgesPerc(){
        int b=getMaximumLengthWithoutBulges();
        double perc = (b*100)/mirna.length();
        return perc;
    }

    //Départ sans hernie
    public  int getStartLengthWithoutBulges(){
        if (mirnaStruc.replace("(", "").trim().length()==0||
                mirnaStruc.replace(")", "").trim().length()==0){
            return mirnaStruc.length();
        }else {
            int cpt=0;
            char c =mirnaStruc.charAt(0);
            int i=0;
            while (c=='('||c==')'){
                c =mirnaStruc.charAt(i);
                cpt++;
                i++;
            }
            return cpt-1;
        }
    }

    //paires de bases dans le duplexe miRNA-miRNA*
    public  int getBasePairsInDuplexMirnaMirnaStar(){
        int p=0;
        for (int i = 0; i < mirna.length(); i++) {
            if (mirnaStruc.charAt(i)=='('||mirnaStruc.charAt(i)==')'){
                p++;
            }
        }
        return p;
    }

    //Présence d'un 20mer
    public  boolean getPresenceOfPerfect20MerBasePair(){
        if
(mirnaStruc.contains("((((((((((((((((((((")||mirnaStruc.contains("))))))))))))))))))))")) {
            return true;
        } else return false;
    }

    // position de depart du 20mer
    public  int getStartOfPerfect20MerBasePair(){
        if (getPresenceOfPerfect20MerBasePair()) {
            int s=mirnaStruc.indexOf("((((((((((((((((((((");
```

```
        if (s==-1){
            s=mirnaStruc.indexOf(")))))))))))))))))))");
        }
        return s+1;
    } else return -1;
}

// paires de bases sur un 10mer
public  boolean getPresenceOfPerfect10MerBasePair(){
    if (mirnaStruc.contains("((((((((((")||mirnaStruc.contains("))))))))))")) {
        return true;
    } else return false;
}

// depart paires de bases sur un 10mer
public  int getStartOfPerfect10MerBasePair(){
    if (getPresenceOfPerfect10MerBasePair()) {
        int s=mirnaStruc.indexOf("((((((((((");
        if (s==-1){
            s=mirnaStruc.indexOf("))))))))))");
        }
        return s+1;
    } else return -1;
}

    // paires de bases sur un 5mer
public  boolean getPresenceOfPerfect5MerBasePair(){
    if (mirnaStruc.contains("(((((")||mirnaStruc.contains(")))))")) {
        return true;
    } else return false;
}

// départ paires de bases sur un 10mer
public  int getStartOfPerfect5MerBasePair(){
    if (getPresenceOfPerfect5MerBasePair()) {
        int s=mirnaStruc.indexOf("(((((");
        if (s==-1){
            s=mirnaStruc.indexOf(")))))");
        }
        return s+1;
    } else return -1;
}

//Depart d'un 8mer en bases pairées parfait suivit par 95% 12mer bases pairées
public  int getStartOfPerfect8And12MerBasePair(){

    return 0;
}

//Presence de A
public  boolean getPresenceOfA(){
    if (mirna.contains("A")){
        return true;
    } else return false;
}

//Pourcentage de A
public  double getPercOfA(){
    int nt=0;
    for (int i = 0; i < mirna.length(); i++) {
        if (mirna.charAt(i)=='A'){
            nt++;
        }
    }
    double perc = (nt*100)/mirna.length();
    return perc;
}

//Presence de U
public  boolean getPresenceOfU(){
    if (mirna.contains("U")){
        return true;
    } else return false;
}

//Pourcentage de U
```

```
public  double getPercOfU(){
    int nt=0;
    for (int i = 0; i < mirna.length(); i++) {
        if (mirna.charAt(i)=='U'){
            nt++;
        }
    }
    double perc = (nt*100)/mirna.length();
    return perc;
}


//Presence de G
public  boolean getPresenceOfG(){
    if (mirna.contains("G")){
        return true;
    } else return false;
}

//Pourcentage de G
public  double getPercOfG(){
    int nt=0;
    for (int i = 0; i < mirna.length(); i++) {
        if (mirna.charAt(i)=='G'){
            nt++;
        }
    }
    double perc = (nt*100)/mirna.length();
    return perc;
}

//Presence de C
public  boolean getPresenceOfC(){
    if (mirna.contains("C")){
        return true;
    } else return false;
}

//Pourcentage de C
public  double getPercOfC(){
    int nt=0;
    for (int i = 0; i < mirna.length(); i++) {
        if (mirna.charAt(i)=='C'){
            nt++;
        }
    }
    double perc = (nt*100)/mirna.length();
    return perc;
}

//Distance à la boucle terminale
public  int getDistanceFromTerminalLoop(){
    int loopStart=precStruc.lastIndexOf("(");
    int loopEnd=precStruc.indexOf(")");
    String arm=getArm();
    if (arm.equals("5'")){
        return loopStart-(prec.indexOf(mirna)+mirna.length());
    } else if (arm.equals("3'")){
            return (prec.indexOf(mirna)-loopEnd);
        } else {
            if (prec.indexOf(mirna)+mirna.length()>loopStart){
                return loopStart-(prec.indexOf(mirna)+mirna.length());
            } else if (prec.indexOf(mirna)<loopEnd){
                return (prec.indexOf(mirna)-loopEnd);
            }
            return 0;
        }
}

//Distance au depart de l'epingle
public  int getDistanceFromHairpinStart(){
    return prec.indexOf(mirna);
}

//Longueur de la superposition sur la boucle
public  int getLengthOfOverlapInLoop(){
```

```
    if (getDistanceFromTerminalLoop()<0){
        return Math.abs(getDistanceFromTerminalLoop());
    } else return 0;
}

// Nombre de paire de bases dans une fentre de 7nt
public  double getAverageNumberOfPairedBasesInWindow7(){
    ArrayList<Integer> al = new ArrayList<Integer>();
    for (int i = 0; i <= mirna.length()-7; i++) {
        String s = mirnaStruc.substring(i, i+7);
        int pair=0;
        for (int j = 0; j < s.length(); j++) {
            if (s.charAt(j) =='('||s.charAt(j) ==')'){
                pair++;
            }
        }
        al.add(pair);
    }
    double tot=0;
    for (Integer i : al) {
        tot=tot+i;
    }
    return (tot/al.size());
}

// Nombre de paire de bases dans une fentre de 5nt
public  double getAverageNumberOfPairedBasesInWindow5(){
    ArrayList<Integer> al = new ArrayList<Integer>();
    for (int i = 0; i <= mirna.length()-5; i++) {
        String s = mirnaStruc.substring(i, i+5);
        int pair=0;
        for (int j = 0; j < s.length(); j++) {
            if (s.charAt(j) =='('||s.charAt(j) ==')'){
                pair++;
            }
        }
        al.add(pair);
    }
    double tot=0;
    for (Integer i : al) {
        tot=tot+i;
    }
    return (tot/al.size());
}

// Nombre de paire de bases dans une fentre de 3nt
public  double getAverageNumberOfPairedBasesInWindow3(){
    ArrayList<Integer> al = new ArrayList<Integer>();
    for (int i = 0; i <= mirna.length()-3; i++) {
        String s = mirnaStruc.substring(i, i+3);
        int pair=0;
        for (int j = 0; j < s.length(); j++) {
            if (s.charAt(j) =='('||s.charAt(j) ==')'){
                pair++;
            }
        }
        al.add(pair);
    }
    double tot=0;
    for (Integer i : al) {
        tot=tot+i;
    }
    return (tot/al.size());
}

//Hernie a la position 2
public  boolean getBulgeAtPosition2(){
    if (mirnaStruc.charAt(1)=='.'){
        return true;
    } else return false;
}

//Hernie a la position -2
public  boolean getBulgeAtPositionMinus2(){
    try {
        int mirnaPos = prec.indexOf(mirna);
```

```
            if (precStruc.charAt(mirnaPos - 2) == '.') {
                return true;
            } else {
                return false;
            }
        } catch (Exception e) { //happening when mirna start at the begining
            return false;
        }
    }

    //Hernie a la position 1
    public  boolean getBulgeAtPosition1(){
        if (mirnaStruc.charAt(0)=='.'){
            return true;
        } else return false;
    }

    //Hernie a la position -1
    public  boolean getBulgeAtPositionMinus1(){
        try {
            int mirnaPos = prec.indexOf(mirna);
            if (precStruc.charAt(mirnaPos - 1) == '.') {
                return true;
            } else {
                return false;
            }
        } catch (Exception e) {
            return false;
        }
    }

    //Nombre de hernies
    public  int getNumberOfBulges(){
        int bulges=0;
        for (int i = 0; i <= mirnaStruc.length()-2; i++) {
            String s = mirnaStruc.substring(i, i+2);
            if (s.equals(".)")||s.equals(".(")){
                bulges++;
            }
        }
        return bulges;
    }

    // Longueur de la plus grande hernies
    public  int getLenghtOfBiggestBulge(){
        int l=0;
        int max=0;
        for (int i = 0; i < mirnaStruc.length(); i++) {
            if (mirnaStruc.charAt(i)=='.'){
                l++;
            }else {
                l=0;
            }
            if (max<l) max=l;
        }
        return max;
    }

    // Longueur du microARN sur la plus grosse hernie en pourcentage
    public  double getLengthBiggestBulgesPerc(){
        int b=getLenghtOfBiggestBulge();
        double perc = (b*100)/mirna.length();
        return perc;
    }

    // recuperation de la region complementaire
    public  String getMirnaStar() {
        if (getMirnaIncludedInLoop()) {
            return "loop";
        }

        String star="";
        int taille = mirna.length();
        int mirnaStart=prec.indexOf(mirna);
        int mirnaEnd=mirnaStart+taille;
        int loopStart=precStruc.lastIndexOf("(");
```

```
int loopEnd=precStruc.indexOf(")");

String arm=null;
try {
    if (precStruc.substring(mirnaStart, mirnaEnd).contains("(")) {
        arm = "5'";
    } else {
        arm = "3'";
    }
} catch (Exception e) {
    //System.err.println("error at "+mirna);
    return "error";
}

 /////////5'
if (arm.equals("5'")){

    // Distance du mirna a la loop effecuté en comptant le nombre de parentheses
    int nbrParentheses=0;
    String intervale = null;
    try {
        intervale = precStruc.substring(mirnaEnd, loopStart);
    } catch (Exception e) {
        intervale="";
    }
    for (char c : intervale.toCharArray()) {
        if (c=='('){
            nbrParentheses++;
        }
    }
    // Determination du départ du mirna star en fonction du nombre de parentheses
    // après la fin de la loop
    int cpt=0;
    int starStart=loopEnd;
    while (cpt!=nbrParentheses){
        if (precStruc.charAt(starStart)==')'){
            cpt++;
            starStart++;
        } else {
            starStart++;
        }
    }

    //recupération du mirna star
    int posOnStar = starStart;
    boolean dec = false;
    for (int i = mirnaEnd; i > mirnaStart; i--) {
        char a = precStruc.charAt(i);
        char b = precStruc.charAt(posOnStar++);
        if (isParenthese(a)&&isParenthese(b)||a==b){
            star+=prec.charAt(posOnStar);
        } else if (a=='.'&&isParenthese(b)){
                i--;
                posOnStar--;
                dec = true;
            } else if (b=='.'&&isParenthese(a)){
                star+=prec.charAt(posOnStar);
            }
    }
    if (dec){
        star+=prec.charAt(posOnStar-1);
    }
}
/////////////////3'
else {

    // Distance du mirna a la loop effecuté en comptant le nombre de parentheses
    int nbrParentheses=0;
    String intervale = null;
    try {
        intervale = precStruc.substring(loopEnd, mirnaStart);
    } catch (Exception e) {
        intervale="";
    }
    for (char c : intervale.toCharArray()) {
        if (c==')'){
```

```
                nbrParentheses++;
            }
        }

        // Determination du départ du mirna star en fonction du nombre de parentheses
        // avant le début de la loop
        int cpt=0;
        int starEnd=loopStart;
        while (cpt!=nbrParentheses){
            if (precStruc.charAt(starEnd)=='('){
                cpt++;
                starEnd--;
            } else {
                starEnd--;
            }
        }

        //recupération du mirna star
        int posOnStar = starEnd;
        boolean dec = false;
        for (int i = mirnaStart; i < mirnaEnd; i++) {
            if (posOnStar>=0) {
                char a = precStruc.charAt(i);
                char b = precStruc.charAt(posOnStar);
                if (isParenthese(a) && isParenthese(b) || a == b) {
                    star = prec.charAt(posOnStar) + star;
                    posOnStar--;
                } else if (a == '.' && isParenthese(b)) {
                    i++;
                    dec = true;
                } else if (b == '.' && isParenthese(a)) {
                    star = prec.charAt(posOnStar) + star;
                    i--;
                    posOnStar--;
                }
            }
        }
        if (dec){
            star=prec.charAt(posOnStar)+star;
        }
    }
    //System.out.println(star);
    return star;
}

public  String getArm(){
    int taille = mirna.length();
    int mirnaStart=prec.indexOf(mirna);
    int mirnaEnd=mirnaStart+taille;
    if (getMirnaIncludedInLoop()) {
        return "loop";
    }
    String arm=null;
    try {
        if (precStruc.substring(mirnaStart, mirnaEnd).contains("(")) {
            arm = "5'";
        } else {
            arm = "3'";
        }
    } catch (Exception e) {
        return "error";
    }
    return arm;
}

public  boolean isParenthese(char a){
    if (a=='('||a==')') return true; else return false;
}


 /**
 * Vérifie que le miRNA n'est pas complètement inclu dans la loop
 * @param prec
 * @param struct
 * @param mirna
 * @return true si le miRNA est dans la loop
```

```
     */
    public  boolean getMirnaIncludedInLoop(){
        try {
            int start = prec.indexOf(mirna);
            int end = start + mirna.length();
            if (end > prec.length()||start==-1) {
                return false;
            }
            if (precStruc.substring(start, end).contains("(")&&precStruc.substring(start,
end).contains(")")) {
                return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }


    public boolean isMultipleLoop() {
        int firstclosePar=precStruc.indexOf(")");
        if (precStruc.substring(firstclosePar).contains("(")){
            return true;
        } else return false;
    }

    @Override
    public String toString(){
        String features="";
        try {
            features = ""
                    + getLength() + ","
                    + getMFE() + ","
                    + df.format(getGCperc()).replace(",", ".") + ","
                    + getMaximumLengthWithoutBulges() + ","
                    + df.format(getMaximumLengthWithoutBulgesPerc()).replace(",", ".") + ","
                    + getStartLengthWithoutBulges() + ","
                    + getBasePairsInDuplexMirnaMirnaStar() + ","
                    + getPresenceOfPerfect20MerBasePair() + ","
                    + getStartOfPerfect20MerBasePair() + ","
                    + getPresenceOfPerfect10MerBasePair() + ","
                    + getStartOfPerfect10MerBasePair() + ","
                    + getPresenceOfPerfect5MerBasePair() + ","
                    + getStartOfPerfect5MerBasePair() + ","
                    + getPresenceOfA() + ","
                    + getPresenceOfU() + ","
                    + getPresenceOfG() + ","
                    + getPresenceOfC() + ","
                    + df.format(getPercOfA()).replace(",", ".") + ","
                    + df.format(getPercOfU()).replace(",", ".") + ","
                    + df.format(getPercOfG()).replace(",", ".") + ","
                    + df.format(getPercOfC()).replace(",", ".") + ","
                    + getDistanceFromTerminalLoop() + ","
                    + getDistanceFromHairpinStart() + ","
                    + getMirnaIncludedInLoop() + ","
                    + getLengthOfOverlapInLoop() + ","
                    + df.format(getAverageNumberOfPairedBasesInWindow7()).replace(",", ".") + ","
                    + df.format(getAverageNumberOfPairedBasesInWindow5()).replace(",", ".") + ","
                    + df.format(getAverageNumberOfPairedBasesInWindow3()).replace(",", ".") + ","
                    + getBulgeAtPosition2() + ","
                    + getBulgeAtPositionMinus2() + ","
                    + getBulgeAtPosition1() + ","
                    + getBulgeAtPositionMinus1() + ","
                    + getNumberOfBulges() + ","
                    + getLenghtOfBiggestBulge() + ","
                    + df.format(getLengthBiggestBulgesPerc()).replace(",", ".") + ","
                    + positive;
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println(mirna+" "+prec+" "+precStruc);
        }
        return features;
    }


    public String toStringBestAttributes(){
```

```
    String features="";
    try {
        features = ""
                + getDistanceFromTerminalLoop() + ","
                + getLengthOfOverlapInLoop() + ","
                + df.format(getAverageNumberOfPairedBasesInWindow3()).replace(",", ".") + ","
                + getBasePairsInDuplexMirnaMirnaStar() + ","
                + getLengthBiggestBulgesPerc() + ","
                + df.format(getAverageNumberOfPairedBasesInWindow5()).replace(",", ".") + ","
                + getLenghtOfBiggestBulge() + ","
                + df.format(getAverageNumberOfPairedBasesInWindow7()).replace(",", ".") + ","
                + getMirnaIncludedInLoop() + ","
                + getDistanceFromHairpinStart() + ","
                + getStartOfPerfect5MerBasePair() + ","
                + getMaximumLengthWithoutBulges() + ","
                + getMaximumLengthWithoutBulgesPerc() + ","
                + getMFE() + ","
                + positive;
    ) catch (Exception e) {
        e.printStackTrace();
        System.out.println(mirna+" "+prec+" "+precStruc);
    }
    return features;
}

     /**
     * Execute windows command with cmd
     * @param cmd
     */
public static String executeWindowsCommand(String cmd) {

    System.out.println(cmd);
    Runtime runtime = Runtime.getRuntime();
    String output="";
    try{
        //Process ps=pb.start();
        Process ps=runtime.exec("cmd /L "+cmd);

        BufferedReader bri=new BufferedReader(new InputStreamReader(ps.getInputStream()));
        String line="";
        while ((line=bri.readLine())!=null) {
           System.out.println(line);
           output=output+line+"\n";
       }
       ps.waitFor();
    }
    catch(Exception e){
        e.printStackTrace();
    }
    return output;
}



public static String executeLinuxCommand (String cmd){

    try {
        ProcessBuilder pb = new ProcessBuilder("bash", "-c", cmd);
        pb.redirectErrorStream(true); // use this to capture messages sent to stderr
        Process shell = pb.start();
        InputStream is = shell.getInputStream(); // this captures the output from the command
        int shellExitStatus = shell.waitFor();

        StringBuilder response = new StringBuilder();
          int value = 0;
          boolean active = true;
          while (active) {
                value = is.read();
                if (value == -1) {
                    throw new IOException("End of Stream");
                } else if (value != '\n') {
                    response.append((char) value);
                    continue;
                } else {
```

```
                    active = false;
                }
            }
            return response.toString();

        } catch (Exception e) {
            return "";
        }
    }
}
```

Feature selection. Ranker output

(Features in red were discarded)

weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
10 folds cross validation

average merit	average rank	attribute
0.226 +- 0.001	1 +- 0	22 DistanceFromTerminalLoop
0.211 +- 0.001	2 +- 0	25 LengthOfOverlapInLoop
0.199 +- 0.001	3 +- 0	28 AverageNumberOfPairedBasesInWindow3
0.187 +- 0.001	4 +- 0	7 BasePairsInDuplexMirnaMirnaStar
0.186 +- 0.001	5 +- 0	35 LengthBiggestBulgesPerc
0.184 +- 0.001	6 +- 0	27 AverageNumberOfPairedBasesInWindow5
0.182 +- 0.001	7 +- 0	34 LenghtOfBiggestBulge
0.166 +- 0.001	8 +- 0	26 AverageNumberOfPairedBasesInWindow7
0.107 +- 0.001	9 +- 0	24 MirnaIncludedInLoop
0.086 +- 0.001	10 +- 0	23 DistanceFromHairpinStart
0.074 +- 0.001	11 +- 0	13 StartOfPerfect5MerBasePair
0.07 +- 0.001	12 +- 0	4 MaximumLengthWithoutBulges
0.069 +- 0.001	13 +- 0	5 MaximumLengthWithoutBulgesPerc
0.055 +- 0.001	14 +- 0	2 mfe
0.035 +- 0.001	15.1 +- 0.3	11 StartOfPerfect10MerBasePair
0.034 +- 0.001	15.9 +- 0.3	6 StartLengthWithoutBulges
0.027 +- 0	17.1 +- 0.3	12 PresenceOfPerfect5MerBasePair
0.026 +- 0	17.9 +- 0.3	10 PresenceOfPerfect10MerBasePair
0.023 +- 0.001	19 +- 0	3 GCperc
0.02 +- 0	20 +- 0	29 BulgeAtPosition2
0.015 +- 0	21 +- 0	20 PercOfG
0.013 +- 0	22 +- 0	33 NumberOfBulges
0.008 +- 0	23 +- 0	18 PercOfA
0.007 +- 0	24.1 +- 0.3	19 PercOfU
0.006 +- 0	25.3 +- 0.46	9 StartOfPerfect20MerBasePair
0.006 +- 0	26 +- 1	32 BulgeAtPositionMinus1
0.006 +- 0	26.6 +- 0.49	8 PresenceOfPerfect20MerBasePair
0.003 +- 0	28.2 +- 0.4	21 PercOfC
0.003 +- 0	28.8 +- 0.4	31 BulgeAtPosition1
0.002 +- 0	30 +- 0	30 BulgeAtPositionMinus2
0.001 +- 0	31 +- 0	16 PresenceOfG
0.001 +- 0	32 +- 0	17 PresenceOfC
0 +- 0	33 +- 0	15 PresenceOfU
0 +- 0	34 +- 0	14 PresenceOfA
0 +- 0	35 +- 0	1 length

Testing models

For every tested model, there was a 10-fold cross validation.

```
Relation:     mirnaInPrecursor
Instances:    37742
Attributes:   15
              DistanceFromTerminalLoop
              LengthOfOverlapInLoop
              AverageNumberOfPairedBasesInWindow3
```

```
              BasePairsInDuplexMirnaMirnaStar
              LengthBiggestBulgesPerc
              AverageNumberOfPairedBasesInWindow5
              LenghtOfBiggestBulge
              AverageNumberOfPairedBasesInWindow7
              MirnaIncludedInLoop
              DistanceFromHairpinStart
              StartOfPerfect5MerBasePair
              MaximumLengthWithoutBulges
              MaximumLengthWithoutBulgesPerc
              mfe
              Class
Test mode:10-fold cross-validation
```

Tree J48

```
=== Run information ===

Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2
Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree

Number of Leaves  :    1783

Size of the tree :     3565


Time taken to build model: 13.94 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       28933               76.66   %
Incorrectly Classified Instances      8809               23.34   %
Kappa statistic                          0.5337
Mean absolute error                      0.282
Root mean squared error                  0.4299
Relative absolute error                 56.4186 %
Root relative squared error             85.9944 %
Total Number of Instances            37742

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.813     0.279      0.739     0.813     0.775      0.796    true
                 0.721     0.187      0.799     0.721     0.758      0.796    false
Weighted Avg.    0.767     0.232      0.77      0.767     0.766      0.796

=== Confusion Matrix ===

     a     b   <-- classified as
 15143  3472 |     a = true
  5337 13790 |     b = false

=== Re-evaluation on control set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances       10257               80.2645 %
```

```
Incorrectly Classified Instances      2522               19.7355 %
Kappa statistic                          0.6053
Mean absolute error                      0.2597
Root mean squared error                  0.39
Total Number of Instances            12779

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.923     0.318      0.744     0.923     0.824      0.838    true
                 0.682     0.077      0.898     0.682     0.776      0.838    false
Weighted Avg.    0.803     0.197      0.821     0.803     0.8        0.838

=== Confusion Matrix ===

    a    b   <-- classified as
 5896  493 |    a = true
 2029 4361 |    b = false

=== Re-evaluation on Prediction set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances        2302               26.5268 %
Incorrectly Classified Instances      6376               73.4732 %
Kappa statistic                          0
Mean absolute error                      0.7437
Root mean squared error                  0.8195
Total Number of Instances             8678

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.265     0          1         0.265     0.419      ?        true
                 0         0.735      0         0         0          ?        false
Weighted Avg.    0.265     0          1         0.265     0.419      0

=== Confusion Matrix ===

    a    b   <-- classified as
 2302 6376 |    a = true
    0    0 |    b = false
```

SMO

```
=== Run information ===

Scheme:weka.classifiers.functions.SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K
   "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

SMO

Kernel used:
  Linear Kernel: K(x,y) = <x,y>


Time taken to build model: 40.27 seconds
```

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       27675               73.3268 %
Incorrectly Classified Instances     10067               26.6732 %
Kappa statistic                          0.4692
Mean absolute error                      0.2667
Root mean squared error                  0.5165
Relative absolute error                 53.3562 %
Root relative squared error            103.3017 %
Total Number of Instances            37742

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.921     0.45       0.666     0.921     0.773      0.736    true
                 0.55      0.079      0.878     0.55      0.676      0.736    false
Weighted Avg.    0.733     0.262      0.773     0.733     0.724      0.736

=== Confusion Matrix ===

     a     b   <-- classified as
 17151  1464 |     a = true
  8603 10524 |     b = false

=== Re-evaluation on control set ===

User supplied control set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances        9354               73.1982 %
Incorrectly Classified Instances      3425               26.8018 %
Kappa statistic                          0.464
Mean absolute error                      0.268
Root mean squared error                  0.5177
Total Number of Instances            12779

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.955     0.491      0.66      0.955     0.781      0.732    true
                 0.509     0.045      0.918     0.509     0.655      0.732    false
Weighted Avg.    0.732     0.268      0.789     0.732     0.718      0.732

=== Confusion Matrix ===

    a    b   <-- classified as
 6099  290 |    a = true
 3135 3255 |    b = false


=== Re-evaluation on prediction set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances        1781               20.5232 %
Incorrectly Classified Instances      6897               79.4768 %
```

```
Kappa statistic                          0
Mean absolute error                      0.7948
Root mean squared error                  0.8915
Total Number of Instances             8678

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.205     0          1         0.205     0.341      ?        true
                 0         0.795      0         0         0          ?        false
Weighted Avg.    0.205     0          1         0.205     0.341      0

=== Confusion Matrix ===

   a    b   <-- classified as
1781 6897 |    a = true
   0    0 |    b = false
```

NeuralNets

```
=== Run information ===

Scheme:weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0
   -E 20 -H a
Test mode:10-fold cross-validation

=== Classifier model (full training set) ===


Time taken to build model: 164.55 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       28528               75.5869 %
Incorrectly Classified Instances      9214               24.4131 %
Kappa statistic                          0.5129
Mean absolute error                      0.3237
Root mean squared error                  0.4051
Relative absolute error                 64.7519 %
Root relative squared error             81.0257 %
Total Number of Instances            37742

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.851     0.337      0.711     0.851     0.775      0.823    true
                 0.663     0.149      0.821     0.663     0.734      0.823    false
Weighted Avg.    0.756     0.242      0.767     0.756     0.754      0.823

=== Confusion Matrix ===

     a     b   <-- classified as
 15844  2771 |     a = true
  6443 12684 |     b = false

=== Re-evaluation on control set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances        9694               75.8588 %
Incorrectly Classified Instances      3085               24.1412 %
```

```
Kappa statistic                          0.5172
Mean absolute error                      0.3183
Root mean squared error                  0.402
Total Number of Instances            12779

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.856    0.338      0.717     0.856     0.78       0.825    true
                 0.662    0.144      0.821     0.662     0.733      0.825    false
Weighted Avg.    0.759    0.241      0.769     0.759     0.756      0.825

=== Confusion Matrix ===

    a    b   <-- classified as
 5467  922 |    a = true
 2163 4227 |    b = false

=== Re-evaluation on test set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances         969               11.1662 %
Incorrectly Classified Instances      7709               88.8338 %
Kappa statistic                          0
Mean absolute error                      0.8269
Root mean squared error                  0.8539
Total Number of Instances             8678

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.112    0          1         0.112     0.201      ?         true
                 0        0.888      0         0         0          ?         false
Weighted Avg.    0.112    0          1         0.112     0.201      0

=== Confusion Matrix ===

    a    b   <-- classified as
  969 7709 |    a = true
    0    0 |    b = false
```

Logistic regression

```
=== Run information ===

Scheme:weka.classifiers.functions.Logistic -R 1.0E-8 -M -1
Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

Time taken to build model: 2.5 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       27795               73.6447 %
Incorrectly Classified Instances      9947               26.3553 %
Kappa statistic                          0.4747
Mean absolute error                      0.3452
```

```
Root mean squared error                  0.4148
Relative absolute error                 69.0495 %
Root relative squared error             82.9626 %
Total Number of Instances            37742

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.868     0.392      0.683     0.868      0.765      0.803    true
                 0.608     0.132      0.826     0.608      0.7        0.803    false
Weighted Avg.    0.736     0.26       0.756     0.736      0.732      0.803

=== Confusion Matrix ===

     a     b   <-- classified as
 16163  2452 |     a = true
  7495 11632 |     b = false

=== Re-evaluation on control set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances        9495               74.3016 %
Incorrectly Classified Instances      3284               25.6984 %
Kappa statistic                          0.486
Mean absolute error                      0.3468
Root mean squared error                  0.4143
Total Number of Instances            12779

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.919     0.433      0.68      0.919      0.781      0.812    true
                 0.567     0.081      0.875     0.567      0.688      0.812    false
Weighted Avg.    0.743     0.257      0.777     0.743      0.735      0.812

=== Confusion Matrix ===

    a    b   <-- classified as
 5871  518 |    a = true
 2766 3624 |    b = false

=== Re-evaluation on prediction set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances        1557               17.9419 %
Incorrectly Classified Instances      7121               82.0581 %
Kappa statistic                          0
Mean absolute error                      0.7607
Root mean squared error                  0.7958
Total Number of Instances             8678

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
```

```
                 0.179     0          1          0.179     0.304     ?         true
                 0         0.821      0          0         0         ?         false
Weighted Avg.    0.179     0          1          0.179     0.304     0

=== Confusion Matrix ===

    a    b   <-- classified as
 1557 7121 |    a = true
    0    0 |    b = false
```

J48 with Adaboost

=== Run information ===

```
Scheme:weka.classifiers.meta.AdaBoostM1     -P     100     -S     1     -I     10     -W
   weka.classifiers.trees.J48 -- -C 0.25 -M 2
Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

AdaBoostM1: Base classifiers and their weights:

J48 pruned tree
------------------

Number of Leaves  :    1097

Size of the tree :     2193


Weight: 0.65

Number of performed Iterations: 10


Time taken to build model: 168.09 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       29042               76.9488 %
Incorrectly Classified Instances      8700               23.0512 %
Kappa statistic                          0.5392
Mean absolute error                      0.2353
Root mean squared error                  0.4509
Relative absolute error                 47.0758 %
Root relative squared error             90.1948 %
Total Number of Instances            37742

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.789     0.25       0.755     0.789     0.772      0.85     true
                 0.75      0.211      0.785     0.75      0.767      0.85     false
Weighted Avg.    0.769     0.23       0.77      0.769     0.769      0.85

=== Confusion Matrix ===

     a     b   <-- classified as
 14689  3926 |     a = true
  4774 14353 |     b = false

=== Re-evaluation on control set ===

User supplied test set
```

```
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances       10949               85.6796 %
Incorrectly Classified Instances      1830               14.3204 %
Kappa statistic                          0.7136
Mean absolute error                      0.1469
Root mean squared error                  0.3545
Total Number of Instances            12779

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.991     0.278      0.781     0.991     0.874      0.925    true
                 0.722     0.009      0.988     0.722     0.835      0.925    false
Weighted Avg.    0.857     0.143      0.885     0.857     0.854      0.925

=== Confusion Matrix ===

    a    b   <-- classified as
 6334   55 |    a = true
 1775 4615 |    b = false

=== Re-evaluation on prediction set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances        1673               19.2786 %
Incorrectly Classified Instances      7005               80.7214 %
Kappa statistic                          0
Mean absolute error                      0.7998
Root mean squared error                  0.8743
Total Number of Instances             8678

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.193     0          1         0.193     0.323      ?        true
                 0         0.807      0         0         0          ?        false
Weighted Avg.    0.193     0          1         0.193     0.323      0

=== Confusion Matrix ===

    a    b   <-- classified as
 1673 7005 |    a = true
    0    0 |    b = false
```

JRip

```
=== Run information ===

Scheme:weka.classifiers.rules.JRip -F 10 -N 2.0 -O 2 -S 1

=== Classifier model (full training set) ===

JRIP rules:
===========

Number of Rules : 34
```

```
Time taken to build model: 123.83 seconds

=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances       29615               78.467  %
Incorrectly Classified Instances      8127               21.533  %
Kappa statistic                          0.5704
Mean absolute error                      0.3254
Root mean squared error                  0.4033
Relative absolute error                 65.0836 %
Root relative squared error             80.6744 %
Total Number of Instances            37742

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.882     0.31       0.735     0.882     0.802      0.812    true
                 0.69      0.118      0.857     0.69      0.765      0.812    false
Weighted Avg.    0.785     0.213      0.797     0.785     0.783      0.812

=== Confusion Matrix ===

     a     b   <-- classified as
 16411  2204 |     a = true
  5923 13204 |     b = false

=== Re-evaluation on control set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances        9750               76.297  %
Incorrectly Classified Instances      3029               23.703  %
Kappa statistic                          0.526
Mean absolute error                      0.3408
Root mean squared error                  0.417
Total Number of Instances            12779

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.897     0.371      0.708     0.897     0.791      0.789    true
                 0.629     0.103      0.859     0.629     0.726      0.789    false
Weighted Avg.    0.763     0.237      0.783     0.763     0.759      0.789

=== Confusion Matrix ===

    a    b   <-- classified as
 5728  661 |    a = true
 2368 4022 |    b = false

=== Re-evaluation on prediction set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===
```

```
Correctly Classified Instances        1737               20.0161 %
Incorrectly Classified Instances      6941               79.9839 %
Kappa statistic                          0
Mean absolute error                      0.751
Root mean squared error                  0.7807
Total Number of Instances             8678

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.2       0          1         0.2       0.334      ?        true
                 0         0.8        0         0         0          ?        false
Weighted Avg.    0.2       0          1         0.2       0.334      0

=== Confusion Matrix ===

    a    b   <-- classified as
 1737 6941 |    a = true
    0    0 |    b = false
```

SVM, Linear kernel

```
=== Run information ===
Scheme:weka.classifiers.functions.LibSVM -S 0 -K 0 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0
   -C 1.0 -E 0.0010 -P 0.1

=== Classifier model (full training set) ===

LibSVM wrapper, original code by Yasser EL-Manzalawy (= WLSVM)

Time taken to build model: 6561.18 seconds

=== Evaluation on training set ===
=== Summary ===

Correctly Classified Instances       27269               72.2511 %
Incorrectly Classified Instances     10473               27.7489 %
Kappa statistic                          0.4485
Mean absolute error                      0.2775
Root mean squared error                  0.5268
Relative absolute error                 55.5081 %
Root relative squared error            105.3642 %
Total Number of Instances            37742

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.962     0.51       0.647     0.962     0.774      0.726    true
                 0.49      0.038      0.929     0.49      0.641      0.726    false
Weighted Avg.    0.723     0.271      0.79      0.723     0.707      0.726

=== Confusion Matrix ===

     a     b   <-- classified as
 17902   713 |     a = true
  9760  9367 |     b = false


=== Re-evaluation on control set ===

User supplied test set
Relation:     mirnaInPrecursor
```

```
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances        9058               70.8819 %
Incorrectly Classified Instances      3721               29.1181 %
Kappa statistic                          0.4177
Mean absolute error                      0.2912
Root mean squared error                  0.5396
Total Number of Instances            12779

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.981     0.564      0.635     0.981     0.771      0.709    true
                 0.436     0.019      0.959     0.436     0.6        0.709    false
Weighted Avg.    0.709     0.291      0.797     0.709     0.685      0.709

=== Confusion Matrix ===

    a    b   <-- classified as
 6270  119 |    a = true
 3602 2788 |    b = false

=== Re-evaluation on Prediction set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances        2782               32.0581 %
Incorrectly Classified Instances      5896               67.9419 %
Kappa statistic                          0
Mean absolute error                      0.6794
Root mean squared error                  0.8243
Total Number of Instances             8678

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.321     0          1         0.321     0.486      ?        true
                 0         0.679      0         0         0          ?        false
Weighted Avg.    0.321     0          1         0.321     0.486      0

=== Confusion Matrix ===

    a    b   <-- classified as
 2782 5896 |    a = true
    0    0 |    b = false
```

SVM, Radial basis kernel

```
=== Run information ===

Scheme:weka.classifiers.functions.LibSVM -S 0 -K 2 -D 3 -G 0.0 -R 0.0 -N 0.5 -M 40.0
   -C 1.0 -E 0.0010 -P 0.1

=== Classifier model (full training set) ===

LibSVM wrapper, original code by Yasser EL-Manzalawy (= WLSVM)
```

```
Time taken to build model: 1417.79 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       29257               77.5184 %
Incorrectly Classified Instances      8485               22.4816 %
Kappa statistic                          0.5498
Mean absolute error                      0.2248
Root mean squared error                  0.4741
Relative absolute error                 44.9714 %
Root relative squared error             94.8382 %
Total Number of Instances            37742

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.734     0.185      0.794     0.734     0.763      0.775    true
                 0.815     0.266      0.759     0.815     0.786      0.775    false
Weighted Avg.    0.775     0.226      0.776     0.775     0.775      0.775

=== Confusion Matrix ===

     a     b   <-- classified as
 13670  4945 |     a = true
  3540 15587 |     b = false

=== Re-evaluation on control set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===

Correctly Classified Instances       11435               89.4827 %
Incorrectly Classified Instances      1344               10.5173 %
Kappa statistic                          0.7897
Mean absolute error                      0.1052
Root mean squared error                  0.3243
Total Number of Instances            12779

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.986     0.197      0.834     0.986     0.904      0.895    true
                 0.803     0.014      0.983     0.803     0.884      0.895    false
Weighted Avg.    0.895     0.105      0.908     0.895     0.894      0.895

=== Confusion Matrix ===

    a    b   <-- classified as
 6301   88 |    a = true
 1256 5134 |    b = false


=== Re-evaluation on test set ===

User supplied test set
Relation:     mirnaInPrecursor
Instances:     unknown (yet). Reading incrementally
Attributes:   15

=== Summary ===
```

```
Correctly Classified Instances         329               3.7912 %
Incorrectly Classified Instances      8349              96.2088 %
Kappa statistic                          0
Mean absolute error                      0.9621
Root mean squared error                  0.9809
Total Number of Instances             8678

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.038    0          1          0.038    0.073      ?         true
                 0        0.962      0          0        0          ?         false
Weighted Avg.    0.038    0          1          0.038    0.073      0

=== Confusion Matrix ===

   a    b   <-- classified as
 329 8349 |    a = true
   0    0 |    b = false
```

ANNEXE C : RESULTATS COMPLEMENTAIRES ET CODE SOURCE DE L'ETUDE SUR LES MICROARNS CHEZ LE BLE

Pour des raisons pratiques, à la vue de l'étendue des résultats, il est impossible d'ajouter à ce mémoire l'ensemble des résultats obtenus, notamment la liste des microARNs et leurs précurseurs.

Tableau C: Gènes cibles du Tableau 3.12. MicroARNs différentiellement exprimés entre les librairies 8 et 10 et leurs gènes cibles. La première colonne correspond à la séquence du microARN, la deuxième au nombre d'ESTs du blé ciblés, les suivantes sont les processus biologiques, fonction moléculaires, et nom des protéines attribués aux cibles portant une référence protéique.

microARNs	Cibles	GO processus biologique	GO fonction moléculaire	GO nom des protéines
UGGGGUGUAAUCUGCG	2	NA	NA	NA
AAAAGCCAUCGACAAA	33	carbohydrate metabolic process; tricarboxylic acid cycle; malate metabolic process; oxidation-reduction process; protein folding; response to stress	catalytic activity;binding;oxidoreductase activity;malate dehydrogenase activity;oxidoreductase activity, acting on the CH-OH group of donors, NAD or NADP as acceptor;L-malate dehydrogenase activity;ATP binding;iron ion binding;oxidoreductase activity, acting on paired donors, with incorporation or reduction of molecular oxygen, 2-oxoglutarate as one donor, and incorporation of one atom each of oxygen into both donors;unfolded protein binding	Malate dehydrogenase;Heat shock protein 90
GGCCUACAUCGCUGAG	3	NA	NA	NA
ACUGGUUGGAUCAUGCUUCUG	3	apoptosis	ATP binding	Resistance protein RGA1R
UCCAAAUCGUGCUUUU	1	NA	NA	NA
GGGCUUAGCUGCUUUU	1	carbohydrate metabolic process; pentose-phosphate shunt	hydrolase activity;6-phosphogluconolactonase activity	Putative 6-phosphogluconolactonase

UAUGUGCUAACAAAAA	3	response to brassinosteroid stimulus	NA	BLE1 protein
UUUGAUCUAACGAAAA	6	NA	NA	NA
AAAAACACGAGAGUUU	3	NA	DNA binding	NA
UUGUGGAGAUAAUGCAAACCC	14	protein phosphorylation; signal transduction	nucleotide binding;protein kinase activity;protein serine/threonine kinase activity;ATP binding;transferase activity;transferring phosphorus-containing groups;structural molecule activity	MAP kinase kinase kinase Ste11/SteC;Protein kinase domain containing;Pto-like protein kinase B
UUUGAGCUACGUUUUU	3	dephosphorylation; transport; photosynthesis; protein-chromophore linkage; electron transport chain;light harvesting	phosphatase activity;chlorophyll binding;metal ion binding;iron-sulfur cluster binding;4 iron, 4 sulfur cluster binding	Function: NEM1 of S. cerevisiae required for correct nuclear morphology;Photosystem I P700 chlorophyll a apoprotein;Chlorophyll a-b binding protein 2, chloroplastic
UUUGCUAGCACAAAAA	1	regulation of transcription, DNA-dependent	DNA binding	NA
CCGUCUUCGUGUUUUU	1	NA	NA	NA
UAGUUCCUAGCUCAGA	3	cell redox homeostasis; oxidation-reduction process	oxidoreductase activity;flavin adenine dinucleotide binding	Monodehydroascorbate reductase
GAAUGACAUCAGAUUC	3	protein phosphorylation	nucleotide binding;protein kinase activity;protein serine/threonine kinase activity;receptor activity;ATP binding;transferase activity, transferring phosphorus-containing groups	FERONIA receptor-like kinase

UCCGAGUCUGCUGCCU	3	rRNA processing;translation	structural constituent of ribosome	Utp14 protein, expressed;60S ribosomal protein L30
GAAUCUGAUGUCAUUC	1	two-component signal transduction system (phosphorelay);chemotaxis;phosphorylation;peptidyl-histidine phosphorylation	two-component sensor activity;nucleotide binding;protein histidine kinase activity;signal transducer activity;ATP binding;transferase activity;transferring phosphorus-containing groups	Chemotaxis protein CheA
UCCUCUGCUCUACCAACUGAGCUAUCCUGAC	2	NA	NA	ORF45d
AGGCAGCAUCUCUAGAU	11	response to stress	nucleotide binding;ATP binding	Heat shock cognate 70 kDa protein 2
UCCUGAUGCUGAACUU	5	NA	NA	NA
AGGCAGCAUCUCUAGA	10	response to stress	nucleotide binding;ATP binding	Heat shock cognate 70 kDa protein 2, putative, expressed
AGGGUAUCGUGGCCAG	1	NA	acid phosphatase activity	Acid phosphatase-like
UUCCUGAGCAUCAUUC	2	NA	NA	NA
UCAAUACAUAUAUGACAA	8	protein folding;cellular protein metabolic process;protein folding	nucleotide binding;ATP binding;unfolded protein binding	T-complex protein 1 subunit alpha

COMMANDES LOGICIELS

Exemples de commandes pour chacune des étapes clé du processus de prédiction des microARNs.

Cutadapt

```
./cutadapt -c -e 0.12 -a 330201030313112312  --maq
~/_color_reads/1/miRNA_Ble_2_20090727_bcSample1_F3.csfasta
~/_color_reads/1/miRNA_Ble_2_20090727_bcSample1_F3_QV.qual >
lib1.fastq
```

MAQ

```
maq fasta2csfa genome_ble6DB.clean.fasta > genome_ble6DB.clean.csfa

maq fasta2bfa genome_ble6DB.clean.csfa genome_ble6DB.clean.csbfa

maq fasta2bfa genome_ble6DB.clean.fasta genome_ble6DB.clean.bfa

maq fastq2bfq lib1_26.fastq lib1_26.bfq

maq map -c lib1_26.genomeble6db.map genome_ble6DB.clean.csbfa
lib1_26.bfq

maq mapview lib1_26.genomeble6db.map > lib1_26.genomeble6db.mapview
```

RepeatMasker

```
perl /ibrixfs1/Data/mik/tools/repeatMasker/RepeatMasker/RepeatMasker
-pa 6 -s -species wheat rnaObject_allLibs.smallRNAOnly.uni.fasta
```

RNAFold

```
RNAFold -noPS < infile.fasta
```

miPred

```
time perl microRNAcheck_parallel.pl -i
lib1_15.smallRNAs.mapped.folded.forPrediction -d temp -f
lib1_15.smallRNAs.mapped.folded.forPrediction.mipred
```

HHMMiR

```
java -jar /ibrixfs1/Data/mik/tools/hhmmir/ExtractHairpins.jar $f
$f.hairpins 40 35 10 > $f.badhairpins

java -jar HHMMiR.jar taeMLEPos.txt taeMLENeg.txt
tae_precursors.hairpins.folded tae_precursors.hairpins.hhmmir0995
0.995
```

TAPIR

```
export PERL5LIB=/ibrixfs1/Data/mik/tools/ViennaRNA-
1.8.4/:/ibrixfs1/Data/mik/tools/ViennaRNA-
1.8.4/Progs:/ibrixfs1/Data/mik/tools/ViennaRNA-
1.8.4/Perl/:/ibrixfs1/Data/mik/tools/ViennaRNA-
1.8.4/Perl/blib/arch/auto/RNA/

PATH=$PATH:/ibrixfs1/Data/mik/tools/ViennaRNA-
1.8.4:/ibrixfs1/Data/mik/tools/ViennaRNA-
1.8.4/Perl/:/ibrixfs1/Data/mik/tools/tapir/Bio/:/ibrixfs1/Data/mik/t
ools/tapir;\ncd /ibrixfs1/Data/mik/cutadapt/targetgenes/

time perl tapir_fasta --mir_file infile.fasta --target_file
/ibrixfs1/Data/mik/genome_ble6DB.clean.fasta > outfile.tapir
```

BLAST

```
blastn.exe -db databases/PMRD.mirnasT -word_size 11 -query
lib.all.mapview.uni.clean.seq.fasta -out
lib.all.mapview.uni.clean.seq.blastPMRD.txt -outfmt "10 qseqid
sallseqid qstart qend sstart send qseq sseq length pident nident
mismatch gaps evalue"
```

HIERARCHIE DU CODE SOURCE

```
|
+---a_cutadapt
|       ESTsCleaning.java
|       i_cutadaptToFasta.java
|       ii_extractSmallrnasByLength.java
|       iii_getExpression.java
|       iv_adaptExpression.java
|       v_parseDumpFile.java
|       vi_getSequencesPrecursorsFromDump.java
|       vii_getSequencesPrecursorsFromMapview.java
|       viii_extracting.java
|       ix_adaptFolding.java
|
+---b_mapping
|       analyseMapping.java
|       mappingQuality.java
|       getUnmappedReads.java
|
+---d_targetgenes
|       i_parse_TAPIR.java
|       ii_geneOntology.java
|       iii_miRNAsTargetGenesStats.java
|
+---e_filters
|       i_parseRibosomaux.java
|       ii_parseLowComplexityRM.java
|
+---f_analyses
|       checkBLASTs.java
|       checkRNAComplements.java
|       conservedAgainstMirnasDBsItol.java
|       i_mirnasStars.java
|       ii_checkMirnasStarsInCutadapt.java
|       iii_conservedAgainstNCrnas.java|
+---g_families
|       checkNumberFamiliesMiRbase.java
|       getFamiliesFromClustalTree.java
|
\---tools
        i_addMAQResultsToObject.java
        ii_insertPredictionInObjects.java
        iii_addTargetsAndAnalysesToObjects.java
        rnaobject2.java
        tools.java
```

CODES SOURCES

a_cutadapt

ESTsCleaning.java

Ce script a eu pour but de nettoyer l'ensemble des ESTs regroupés de plusieurs bases de données. De nombreux doublons sont apparus et il a été nécessaire de les enlever pour éviter les faux duplicatas lors du mappage et la recherche des génes cibles.

```
/*
 * Remove duplicate sequences
 * concatenate names of duplicate sequences
 * Chosing the longest name and create index of names
 *
*/
package d_targetgenes;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;

/**
 *
 * @author mickael
 */
public class ESTsCleaning {

    public static void main(String args[]){
        File genome = new File("/ibrixfs1/Data/mik/genome_ble6DB.fasta");
        File outfile = new File("/ibrixfs1/Data/mik/genome_ble6DB.clean.fasta");
        File indexfile = new File("/ibrixfs1/Data/mik/genome_ble6DB.index.txt");

        //int est = tools.check_number_contigs(genome);
        int est = 1756788;
        HashMap<String,String> hm = new HashMap<String, String>(est+10000);

        System.out.println("Hashing...");
        try {
            BufferedReader br = new BufferedReader(new FileReader(genome));
            String line = br.readLine();
            while (br.ready()){
                String name="";
                String seq="";
                if (line.startsWith(">")){
                    name = line.substring(1).replace("\t", " ").replaceAll(",", "
").replaceAll(";", " ");
                    seq=br.readLine();
                    line = br.readLine();
                    while(line!=null&&!line.startsWith(">")){
                        seq+=line;
                        line = br.readLine();
                    }
                } else {
                    line=br.readLine();
                }
                if (hm.containsKey(seq)){
                    String tmp = hm.get(seq);
                    tmp+="\t"+name;
                    hm.put(seq, tmp);
                } else {
                    hm.put(seq, name);
                }
            }
            br.close();

            System.out.println("print outfile and index");
            if (outfile.exists()) outfile.delete(); if (indexfile.exists())indexfile.delete();
            PrintWriter pwout = new PrintWriter(new FileWriter(outfile));
```

```
            PrintWriter pwindex = new PrintWriter(new FileWriter(indexfile));
            int cpt=0;
            for (String seq : hm.keySet()) {
                cpt++;
                String list[]=hm.get(seq).split("\t");
                int bestLengthposition=0;

                if (list.length>1){
                    for (int i = 1; i < list.length; i++) {
                        if (list[i].length()>list[bestLengthposition].length()){
                            bestLengthposition=i;
                        }
                    }
                }

                String names = hm.get(seq);
                if (list[bestLengthposition].contains("ABI")&&
                        names.contains("gi|")&&names.split("\t").length==2){
                    if(bestLengthposition==0) {
                        bestLengthposition=1;
                    }
                }

                pwout.println(">"+list[bestLengthposition].replaceAll("  ", " ")
+"\n"+seq.replaceAll("  ", " "));

                pwindex.println(list[bestLengthposition]+"\t"+hm.get(seq));

            }
            pwout.close();
            pwindex.close();
            System.out.println("Number of contigs: "+cpt);
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

}
```

getUnmappedReads.java

Récupère les reads qui n'ont pas mappé contre les ESTs du blé. Sachant que les ESTs ne contiennent que les régions codantes de l'ADN, il manque potentiellement ceux des introns et des régions intergéniques. Un mappage de ces reads contre d'autres espèces proches du blé et ayant un génome complet permettrait d'obtenir d'autres microARNs.

```
/*
 * Get Unmapped reads
 * Execute first:  for i in {1..10}; do for f in $i/*.txt; do awk '{print $2}' $f >
$f.readsList;done;done in /ibrixfs1/Data/mik/cutadapt/objects
 * in order to get mapped reads list
 */
package a_cutadapt;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;
import tools.tools;

/**
 *
 * @author Mickael
 */
public class getUnmappedReads {
    public static HashMap<String,Integer> hmMappedReads;
```

```
    public static void main(String args[]){

        String fmap = "/ibrixfs1/Data/mik/cutadapt/mapping/";
        String fobj = "/ibrixfs1/Data/mik/cutadapt/objects/";
        for (int i = 1; i <= 10; i++) {
            for (int j = 15; j <= 30; j++) {
                System.out.println("lib"+i+"_"+j);
                File fastq = new File (fmap+""+i+"/lib"+i+"_"+j+".fastq");
                File mappedReads = new File (fobj+""+i+"/lib"+i+"_"+j+".txt.readsList");
                File outfile = new File (fmap+""+i+"/lib"+i+"_"+j+".unmapped.fastq");
                addmappedReadsToHashMap(mappedReads);
                getNonmappedReads(fastq,outfile);
            }
        }
    }


    private static void getNonmappedReads(File fastq, File outfile) {

        try {
            BufferedReader br = new BufferedReader(new FileReader(fastq));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile)));

            String line="";
            while (br.ready()){
                line=br.readLine();
                if (!hmMappedReads.containsKey(line.substring(1))){
                    String name=line;
                    String seq=br.readLine();
                    String plus= br.readLine();
                    String qual = br.readLine();
                    pw.println(name+"\n"+seq+"\n"+plus+"\n"+qual);
                } else {
                    br.readLine();br.readLine();br.readLine();
                }

            }

           pw.close();
           br.close();
        } catch (Exception e) {
        }
    }


    private static void addmappedReadsToHashMap(File mappedReads) {
        int countlines = tools.countLines(mappedReads);
        //System.out.println("Adding mappedReads to hashmap");
        hmMappedReads = new HashMap<String, Integer>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(mappedReads));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                hmMappedReads.put(line, 0);
            }
            br.close();
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in addmappedReadsToHashMap at "+i+" on "+countlines);
        }
    }
}
```

i_cutadaptToFasta.java

Converti les données fournies dans des fichiers FASTQ par cutadapt en format MAQ en format FASTA.

```
/*
 * Convert cutadapt data in MAQ format in fasta
 */

package a_cutadapt;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;

/**
 *
 * @author Mickael
 */
public class i_cutadaptToFasta {
    public static int lib=0;
    public static int minLenght=15;
    public static int maxLenght=30;

    public static void main(String args[]){

        for (lib = 1; lib <= 1; lib++) {

            try {
                String f = "/ibrixfs1/Data/mik/cutadapt/mapping/";
                File cutatapfile = new File(f + "lib" + lib + ".fastq");
                File outfile = new File(f + "lib" + lib + ".allLengths.fasta");
                File outfilelenght = new File(f + "lib" + lib + ".lenghts.txt");
                if (outfile.exists())outfile.delete();
                if (outfilelenght.exists()) outfilelenght.delete();

                convertToFasta(cutatapfile, outfile, outfilelenght);
            } catch (Exception e) {

            }
        }

    }


    private static void convertToFasta(File cutatapfile, File outfile, File outfilelenght) {
        System.out.println("Converting lib"+lib);
        int cpt=0;

        int tailles[] = new int[35];
        for (int i : tailles) {
            tailles[i]=0;
        }
        try {
            BufferedReader br = new BufferedReader(new FileReader(cutatapfile));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
            PrintWriter pw2 = new PrintWriter(new BufferedWriter(new FileWriter(outfilelenght,
true)));

            while (br.ready()){
                String name = br.readLine();
                if (!name.startsWith("@")){
                    System.err.println(name);
                }
                String seq = br.readLine();
                String strand = br.readLine();
                if (strand.length()!=1){
                    System.err.println(strand);
                }
                String qual = br.readLine();
                name = name.substring(1,name.indexOf("/"));
                int t = seq.length();
                tailles[t]=tailles[t]+1;
                pw.println(">" + name + "\t" + strand + "\n" + seq);
                cpt++;
```

```
                if (cpt%1000000==0){
                    System.out.print("*");
                    pw.flush();
                }
            }
            System.out.println("");
            for (int i = 0; i < tailles.length; i++) {
                System.out.println(i+"\t"+tailles[i]);
                pw2.println(i+"\t"+tailles[i]);
            }

            System.out.println("Finished");
            pw.close();
            pw2.close();
            br.close();

        } catch (Exception e) {
            System.out.println("lib "+lib+ " not found");
        }
    }

    private static void getLentghs(File infile, File outfile) {
        System.out.println("Processing lenghts...");
        int tailles[] = new int[35];
        for (int i : tailles) {
            tailles[i]=0;
        }
        int cpt=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
            String line = "";
            while (br.ready()) {
                line = br.readLine();
                if (line.startsWith("A")||line.startsWith("T")
                        ||line.startsWith("G")||line.startsWith("C")){
                    int t = line.length();
                    tailles[t]=tailles[t]+1;
                    cpt++;
                    if (cpt%1000000==0)System.out.print("*");
                }

            }
            System.out.println("total lines : "+cpt);
            br.close();

            for (int i = 0; i < tailles.length; i++) {
                System.out.println(i+"\t"+tailles[i]);
                pw.println(i+"\t"+tailles[i]);
            }
            br.close();
            pw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("finished!");

    }

}
```

ii_extractSmallrnasByLength.java

MAQ a la particularité de ne pouvoir traiter des fichiers qui contiennent des séquences de taille unique. A partir des fichiers fastq créés par cutadapt des fichiers sont créés selon la taille des séquences. Au final il y aura dix dossiers, pour dix librairies, et dans chacun d'entre eux seize fichiers, contenant les séquences de 15 à 30.

```
/*
 * A partir des fichiers fastq créés par cutadapt,des fichiers sont créés
 * selon la taille des séquences produites.
 * Il y a un dossier par librairie
 * Créer les dossiers à la main
 */
```

```
package a_cutadapt;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;

/**
 *
 * @author Mickael
 */
public class ii_extractSmallrnasByLength {

    public static int lib=0;
    public static int minLenght=15;
    public static int maxLenght=30;

    public static void main(String args[]){
//        fastq
        for (lib = 1; lib <= 10; lib++) {
            try {
                String f = "H:\\mik\\_version3\\i_mapping\\cutadapt\\";
                File infile = new File(f + "lib" + lib + ".fastq");
                File outfolder = new File(f + "\\" + lib);
                outfolder.mkdir();
                extractSmallrnasByLibsFastq(infile, outfolder);
            } catch (Exception e) {

            }
        }

//        //fasta
//        for (lib = 1; lib <= 10; lib++) {
//            try {
//                String f = "H:\\mik\\_version3\\i_mapping\\cutadapt\\";
//                File infile = new File(f + "lib" + lib + ".fasta");
//                File outfolder = new File(f + "\\fasta\\" + lib);
//                outfolder.mkdir();
//                extractmirnasByLibsFasta(infile, outfolder);
//            } catch (Exception e) {
//
//            }
//        }
    }

    /**
     * S'occupe de splitter les fichiers fastq par taille
     * @param infile
     * @param outfolder
     */
    public static void extractSmallrnasByLibsFastq(File infile, File outfolder) {
        System.out.println("Extracting lib"+lib);
        int cpt=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            PrintWriter pwtab[]= new PrintWriter[16];
            for (int i = 0; i <= 15; i++) {
                int n = i+15;
                File f = new File(outfolder+File.separator+"lib"+lib+"_"+n+".fastq");
                if (f.exists()) f.delete();
                pwtab[i]=new PrintWriter(new BufferedWriter(new FileWriter(f, true)));
            }


            while (br.ready()) {
                 String name = br.readLine();
                if (!name.startsWith("@")){
                    System.err.println(name);
                }
                String seq = br.readLine();
                String sens = br.readLine();
                if (sens.length()!=1){
                    System.err.println(sens);
```

```
                }
                String qual = br.readLine();

                int t = seq.length();
                if (t>=minLenght&&t<=maxLenght) {
                    pwtab[t - 15].println(name + "\n" + seq + "\n" + sens + "\n" + qual);
                    cpt++;
                    if (cpt % 100000 == 0) {
                        System.out.print("*");
                        for (PrintWriter p : pwtab) {
                            p.flush();
                        }
                    }
                }

                cpt++;
            }
            System.out.println("Finished");
            for (PrintWriter p : pwtab) {
                p.close();
            }
            br.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    /**
     * S'occupe de splitter les fichiers fasta par taille
     * @param infile
     * @param outfolder
     */
    public static void extractSmallrnasByLibsFasta(File infile, File outfolder) {
        System.out.println("Extracting lib"+lib);
        int cpt=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            PrintWriter pwtab[]= new PrintWriter[16];
            for (int i = 0; i <= 15; i++) {
                int n = i+15;
                File f = new File(outfolder+File.separator+"lib"+lib+"_"+n+".fasta");
                if (f.exists()) f.delete();
                pwtab[i]=new PrintWriter(new BufferedWriter(new FileWriter(f, true)));
            }

            String line = "";
            while (br.ready()) {
                line = br.readLine();

                if (line.startsWith(">")) {
                    String name = line;
                    String smallrna= br.readLine();
                    int l = smallrna.length();
                    pwtab[l-15].println(name+"\n"+smallrna);
                    cpt++;
                    if (cpt%100000==0) {
                        System.out.print("*");
                        for (PrintWriter p : pwtab) {
                            p.flush();
                        }
                    }
                }
            }
            System.out.println("Finished");
            for (PrintWriter p : pwtab) {
                p.close();
            }
            br.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

iii_getExpression.java

Obtient le niveau d'expression de chaque code couleur unique dans toutes les librairies. Ce niveau d'expression renseignera sur l'abondance d'un petit ARN selon le stress induit à la plante.

```
/**
 * Permet d'obtenir le niveau d'expression des reads dans toutes les
 * librairies
 */
package a_cutadapt;


import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;


/**
 *
 * @author Mickael
 */
public class iii_getExpression {

    public static void main (String args[]){
//        File sequences =new File (args[0]);
//        File lib_folder = new File (args[1]);

        File sequences = new File("/ibrixfs1/Data/mik/cutadapt/fasta/all_lib.smallrna.uni.txt");
        File lib_folder = new File("/ibrixfs1/Data/mik/cutadapt/");

        getExpression(sequences, lib_folder);
        sequences.renameTo(new File
("/ibrixfs1/Data/mik/cutadapt/fasta/all_lib.smallrna.uni.expression.txt"));
    }


     /**
     * Modifie les noms en identifiant la présence du read dans chaque librairie
     * et par la même occasion le profil d'expression (combien de fois le reads est
     * trouvé dans chaque librairie)
     * @param sequences reads file path
     * @param lib_folder libraries folder path
     */
    public static void getExpression(File sequences, File lib_folder) {
        try {
            for (int i = 1; i <= 10; i++) {

                File fastaFile= new File(lib_folder.toString()+File.separator
                        +File.separator+"lib"+i+".fasta");
                //check number of reads in the library
                int lib_size = tools.tools.check_number_contigs(fastaFile);

                //Add lib in a hasmap
                HashMap<Integer,Integer> hm = new HashMap<Integer,Integer>(lib_size,10000);
                BufferedReader br_libs=new BufferedReader(new FileReader(fastaFile));
                String line = "";
                int m = 0;
                System.out.println("Put library lib"+i+" in Hashmap...");
                try {
                    while (br_libs.ready()) {
                        line = br_libs.readLine();
                        if (line.startsWith(">")) {
                            line = br_libs.readLine();
                            Integer hash=line.hashCode();
                            Object obj = hm.get(hash);
                            if (obj!=null) {
                                Integer tmp=hm.get(hash);
                                int k=tmp+1;
                                hm.put(hash, k);
                            } else {
```

```
                                hm.put(hash, 1);
                            }
                            m++;
                            if (m % 1000000 == 0) {
                                System.out.print("*");
                            }
                        }
                    }
                    br_libs.close();
                } catch (Exception ex) {
                    ex.printStackTrace();
                }

                //search smallRNA in libs (stored the hashmap)
                File outfile = new File(sequences.toString()+".named");
                PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile,
true)));
                BufferedReader br_fasta=new BufferedReader(new FileReader(sequences));

                int cpt=0;
                System.out.println("\nSearching expression profile for lib"+i+" in HashMap...");
                line = "";
                while(br_fasta.ready()){
                    cpt++;
                    line = br_fasta.readLine();
                    String sequence = line.split("\t")[0];
                    int hits = 0;
                    try {
                        hits = hm.get(sequence.hashCode());
                    } catch (Exception e) {
                        hits=0;
                    }
                    if (i==1){
                        pw.println(sequence+"\t"+hits};
                    }else{
                        pw.println(line+","+hits);
                    }
                    if (cpt%1000000==0){
                        System.out.print("*");
                        pw.flush();
                    }
                }
                pw.close();
                br_fasta.close();
                do {
                    sequences.delete();
                } while (sequences.exists());
                outfile.renameTo(sequences);

                System.out.println("Library "+i+" finished \n");

            }
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

}
```

iv_adaptExpression.java

Ce script ne sert qu'à changer des virgules en tabulations et à calculer le total dans toutes les librairies.

```
/*
 * Modifie le fichier de getExpression en calculant un total et en remplacant les
 * virgules par des tab. Utile pour l'analyse differentielle
 */

package a_cutadapt;
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.BufferedWriter;
import java.io.FileReader;

/**
 *
 * @author Mickael
 */
public class iv_adaptExpression {

    public static void main(String args[]){
        String f = "H:\\mik\\_version3\\i_mapping\\cutadapt\\fasta\\";

        File infile = new File(f+"all_lib.smallrna.uni.expression.txt");
        File outfile = new File(f+"all_lib.smallrna.uni.expression.tab.tot.sup5.txt");

        process(infile,outfile);
    }

    private static void process(File infile, File outfile) {
        System.out.println("Parsing file...");
        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
            String line = "";
            int cpt=0;
            while (br.ready()){
                line = br.readLine();
                String name = line.split("\t")[0];
                String exp[] = line.split("\t")[1].split(",");

                int total=0;
                String expr="";
                for (int j = 0; j < exp.length; j++) {
                    total+=Integer.valueOf(exp[j]);
                    expr+=Integer.valueOf(exp[j])+"\t";
                }
                if (total>=5) {
                    pw.println(name + "\t" + expr + total);
                }
                if (cpt%100000==0){
                    System.out.print("*");
                    pw.flush();
                }
                cpt++;
            }
            System.out.println("Finished");
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

}
```

v_parseDumpFile.java

MAQ met dans un fichier dump l'ensemble des multiples hits du mapping. Les meilleurs hits sont dans un autre fichier (mapview). La structure d'un fichier dump est très particulière et nécessite d'être réorganisée si on doit l'exploiter. Ce script transforme donc le fichier dump pour le rendre exploitable.

```
/*
 * Lit le fichier dump de maq et crée un fichier en sortie
 * output :
 * read name    source  position    strand  mismatchs
 * 2118_992_852/1TA49712_4565     32       -       2
 *
 * Entre 15 et 17 inclus, seuls 0 et 1 mismatchs sont acceptés
```

```
 * Au dessus de 17, 0,1,2,3 mismatchs sont acceptés
 *
 */

package a_cutadapt;

import java.io.File;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;

/**
 *
 * @author Mickael
 */
public class v_parseDumpFile {

    public static void main (String args[]){
        System.out.println("Parse dump file");

        //String f = args[0]; //"H:\\mik\\_version3\\i_mapping\\cutadapt\\";
        String f = "/ibrixfsl/Data/mik/cutadapt/mapping/";

        for (int i = 1; i <= 1; i++) {
            for (int j = 15; j <= 15; j++) {
                File dump = new File(f+i+"/lib"+i+"_"+j+".genomeble6db.dump");
                File outfile = new File(f+i+"/lib"+i+"_"+j+".genomeble6db.dump.out");
                System.out.println("\nlib"+i+"_"+j);
                if (outfile.exists()) outfile.delete();
                parsing(j, dump,outfile);
            }
        }
        //vi_getSequencesPrecursorsFromDump.main(args);
        //vii_getSequencesPrecursorsFromMapview.main(args);
    }

    private static void parsing(int taille, File dump,File outfile) {
        System.out.println("Parsing file "+dump.getName());
        HashMap<Integer,String> hmReadIds = new HashMap<Integer,String>();
        int cpt=1;
        int id=0;
        String name="";
        try {
            BufferedReader br = new BufferedReader(new FileReader(dump));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
            String line="";
            while (br.ready()){
                line = br.readLine();
                if (line.startsWith("R")){
                    id =Integer.valueOf(line.split("\t")[1]);
                    try {
                        name = line.split("\t")[2];
                    } catch (Exception e) {
                        name = "unknown";
                    }
                    hmReadIds.put(id,name);
                }
                if (line.startsWith("B")){
                    String next = br.readLine();
                    while (next.startsWith("A")){
                        String tab[] = next.split("\t");
                        /**
                         * Impression ou non dans le fichier
                         * Selon la taille et le nombre de mismatchs
                         */
                        if (taille<=17) {
                            if (Integer.valueOf(tab[4])<=1) {
                                cpt++;
                                pw.println(
                                        hmReadIds.get(Integer.valueOf(tab[1])) //nom read
                                        + "\t" + line.split("\t")[1] //nom source cible
                                        + "\t" + tab[2] //start
                                        + "\t" + tab[3] //strand
```

```
                                    + "\t" + tab[4] //mismatchs
                                    );
                            }
                        } else {
                            cpt++;
                            pw.println(
                                    hmReadIds.get(Integer.valueOf(tab[1])) //nom read
                                    + "\t" + line.split("\t")[1] //nom source cible
                                    + "\t" + tab[2] //start
                                    + "\t" + tab[3] //strand
                                    + "\t" + tab[4] //mismatchs
                                    );
                        }
                        next = br.readLine();
                    }
                }
                if (cpt%100000==0){
                    System.out.print("*");
                    pw.flush();
                }
            }
            pw.close();
        } catch (Exception e) {
            System.out.println("Error at "+id);
            e.printStackTrace();
        }
    }
}
```

vi_getSequencesPrecursorsFromDump.java

Une fois que le fichier dump a été réorganisé, ce script récupére les séquences dans les ESTs ainsi que leurs précurseurs.

```
/*
 * Récupération des séquences sur le génome de référence à partir du dump
 */

package a_cutadapt;

import java.io.File;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

/**
 *
 * @author Mickael
 */
public class vi_getSequencesPrecursorsFromDump {

    public static void main (String args[]){
        System.out.println("Get Sequences Precursors from dump");
        File genome = new File ("/ibrixfs1/Data/mik/genome_ble6DB.clean.fasta");

        String f = "/ibrixfs1/Data/mik/cutadapt/mapping/";
        for (int i = 1; i <= 10; i++) {
            for (int j = 15; j <= 30; j++) {
                File dump = new File(f+i+"/lib"+i+"_"+j+".genomeble6db.dump.out");
                File outfile = new File(f+i+"/lib"+i+"_"+j+".genomeble6db.dump.out.seq");
                System.out.println("\nlib"+i+"_"+j);
                if (outfile.exists()) outfile.delete();
                getSequences(j,dump,outfile,genome);
            }
        }

    }

    private static void getSequences(int taille, File dump, File outfile, File genome) {
        int cpt=0;
        try {
```

```
BufferedReader br_dump = new BufferedReader(new FileReader(dump));
BufferedReader br_genome = new BufferedReader(new FileReader(genome));
PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
String linedump="";
String linegenome="";
String nomcontig="";
String sequence="";

// stockage du premier contig
linegenome = br_genome.readLine();
if (linegenome.startsWith(">")){
    nomcontig = linegenome;
    linegenome = br_genome.readLine();
    //recuperation de la sequence
    while (!linegenome.startsWith(">")){
        sequence+=linegenome;
        linegenome = br_genome.readLine();
    }
}

while (br_dump.ready()){
    try {
        linedump = br_dump.readLine();
        String s[] = linedump.split("\t");
        /**
         * recherche du nom du contig dans le génome
         * Si la ligne suivante du fichier dump utilise le meme contig
         * alors on passe cette étape
         */
        if (!nomcontig.startsWith(">" + s[1])) {
            while (!linegenome.startsWith(">" + s[1])) {
                linegenome = br_genome.readLine();
            }
            nomcontig = linegenome;
            linegenome = br_genome.readLine();
            sequence = "";
            try {
                while (!linegenome.startsWith(">")) {
                    sequence += linegenome;
                    linegenome = br_genome.readLine();
                }
            } catch (Exception ex) {
                ex.printStackTrace();
                System.err.println("problem at " + nomcontig);
                System.err.println("line " + linegenome);
                System.err.println("cpt " + cpt);
            }
        }

        /**
         * get sequence and precursor
         * MAQ ne renvoie pas la position de départ, donc on est obligé de le
corriger
         */
        int position = Integer.valueOf(s[2]) - (taille);
        String smallRNA = sequence.substring(position, position + taille);

        //Precursor 1
        int start = position - 20;
        if (start < 0)start = 0;

        int end = position + taille + 160;
        if (end > sequence.length())end = sequence.length();

        String precursor1 = sequence.substring(start, end);

        //Precursor 1
        start = position - 160;
        if (start < 0)start = 0;

        end = position + taille + 20;
        if (end > sequence.length())end = sequence.length();

        String precursor2 = sequence.substring(start, end);
```

```
                    // suppression des tab dans le nom
                    String sourcename = nomcontig.replace("\t", " ").replace(",", " ");
                    if (sourcename.contains(" ")) {
                        sourcename = sourcename.substring(sourcename.indexOf(" ") + 1);
                    } else {
                        sourcename = "unknown";
                    }

                    //ecriture avec precurseur 1
                    pw.println(
                            s[0]
                            + "\t" +
                            smallRNA
                            + "\t" + s[3] //strand
                            + "\t" + s[4] //mismatch
                            + "\t" + s[1]
                            + "\t" + sourcename
                            + "\t" + precursor1);

                    //ecriture avec precurseur 2
                    pw.println(
                            s[0]
                            + "\t" +
                            smallRNA
                            + "\t" + s[3]
                            + "\t" + s[4]
                            + "\t" + s[1]
                            + "\t" + sourcename
                            + "\t" + precursor2);

                    //ecriture simple
//                    pw.println(s[0]+"\t"+smallRNA+"\t"+
s[3]+"\t"+s[4]+"\t"+""+"\t"+""+"\t"+precursor1);
//                    pw.println(s[0]+"\t"+smallRNA+"\t"+
s[3]+"\t"+s[4]+"\t"+""+"\t"+""+"\t"+precursor2);


                    if (cpt % 50000 == 0) {
                        System.out.print("*");
                        pw.flush();
                    }
                    cpt++;
                } catch (IOException iOException) {
                } catch (NumberFormatException numberFormatException) {
                }
            }
            br_dump.close();
            br_genome.close();
            pw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

vii_getSequencesPrecursorsFromMapview.java

Le fichier mapview contient les informations sur les meilleurs hits des reads contre les ESTs (génome). Ce script récupère les séquences du séquençage ainsi que les précurseurs correspondants.

```
/*
 * Récupération des séquences sur le génome de référence à partir du mapview
 */

package a_cutadapt;

import java.io.File;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
```

```
/**
 *
 * @author Mickael
 */
public class vii_getSequencesPrecursorsFromMapview {

    public static void main (String args[]){
        System.out.println("Get Sequences Precursors from mapview");
        File genome = new File ("/ibrixfs1/Data/mik/genome_ble6DB.clean.fasta");

        String f = "/ibrixfs1/Data/mik/cutadapt/mapping/";
        for (int i = 1; i <= 10; i++) {
            for (int j = 15; j <= 30; j++) {
                File dump = new File(f+i+"/lib"+i+"_"+j+".genomeble6db.mapview");
                File outfile = new File(f+i+"/lib"+i+"_"+j+".genomeble6db.mapview.seq");
                System.out.println("\nlib"+i+"_"+j);
                if (outfile.exists()) outfile.delete();
                getSequences(j,dump,outfile,genome);
            }
        }
    }

        private static void getSequences(int taille, File dump, File outfile, File genome) {
        int cpt=0;
        try {
            BufferedReader br_map = new BufferedReader(new FileReader(dump));
            BufferedReader br_genome = new BufferedReader(new FileReader(genome));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
            String linemap="";
            String linegenome="";
            String nomcontig="";
            String sequence="";

            // stockage du premier contig
            linegenome = br_genome.readLine();
            if (linegenome.startsWith(">")){
                nomcontig = linegenome;
                linegenome = br_genome.readLine();
                //recuperation de la sequence
                while (!linegenome.startsWith(">")){
                    sequence+=linegenome;
                    linegenome = br_genome.readLine();
                }
            }

            while (br_map.ready()){
                try {
                    linemap = br_map.readLine();
                    String s[] = linemap.split("\t");
                    /**
                     * recherche du nom du contig dans le génome
                     * Si la ligne suivante du fichier dump utilise le meme contig
                     * alors on passe cette étape
                     */
                    if (!nomcontig.startsWith(">" + s[1])) {
                        while (!linegenome.startsWith(">" + s[1])) {
                            linegenome = br_genome.readLine();
                        }
                        nomcontig = linegenome;
                        linegenome = br_genome.readLine();
                        sequence = "";
                        try {
                            while (!linegenome.startsWith(">")) {
                                sequence += linegenome;
                                linegenome = br_genome.readLine();
                            }
                        } catch (Exception ex) {
                            ex.printStackTrace();
                            System.err.println("problem at " + nomcontig);
                            System.err.println("line " + linegenome);
                            System.err.println("cpt " + cpt);
                        }
                    }

                    /**
```

```
                 * get sequence and precursor
                 *
                 *
                 */
                int position = Integer.valueOf(s[2])-1;
                String smallRNA = sequence.substring(position, position + taille);

                //Precursor 1
                int start = position - 20;
                if (start < 0)start = 0;

                int end = position + taille + 160;
                if (end > sequence.length())end = sequence.length();

                String precursor1 = sequence.substring(start, end);

                //Precursor 2
                start = position - 160;
                if (start < 0)start = 0;

                end = position + taille + 20;
                if (end > sequence.length())end = sequence.length();

                String precursor2 = sequence.substring(start, end);

                // suppression des tab dans le nom
                String sourcename = nomcontig.replace("\t", " ").replace(",", " ");
                if (sourcename.contains(" ")) {
                    sourcename = sourcename.substring(sourcename.indexOf(" ") + 1);
                } else {
                    sourcename = "unknown";
                }

                //ecriture avec precurseur 1
                pw.println(
                        linemap+
                        "\t" + smallRNA + //smallRNA extrait
                        "\t" + sourcename + //source name
                        "\t" + precursor1);    //precursor

                //ecriture avec precurseur 2
                pw.println(
                        linemap+
                        "\t" + smallRNA + //smallRNA extrait
                        "\t" + sourcename + //source name
                        "\t" + precursor2);    //precursor


                if (cpt % 50000 == 0) {
                    System.out.print("*");
                    pw.flush();
                }
                cpt++;
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
        br_map.close();
        br_genome.close();
        pw.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

ix_adaptFolding.java

Quelques ESTs contenaient encore des N dans leur séquence, et comme les logiciels de prédiction ne pouvaient les prendre en compte, les séquences furent coupées au niveau des N avant de les replier. Ce script permet de vérifier dans quelle partie de séquence coupée et repliée se trouve le petit ARN séquencé. D'autre part, il recréé la structure du fichier qui n'était plus en format FASTA après le repliement.

```
/*
 * Adapt RNAfold output for prediction
 * Creer le dossier adapted et les sous dossiers 1 à 10 avant
 *
 */

package a_cutadapt;
import java.io.File;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.Vector;
/**
 *
 * @author Mickael
 */
public class ix_adaptFolding {


    public static void main (String args[]){
        System.out.println("Adapt precursors");

        String f = "/ibrixfs1/Data/mik/cutadapt/folding/";
        for (int i = 1; i <= 10; i++) {
            for (int j = 15; j <= 30; j++) {
                File infile = new
File(f+i+"/lib"+i+"_"+j+".genomeble6db.mapped.seq.uni.forFolding.folded");
                File outfile = new
File(f+i+"/lib"+i+"_"+j+".smallRNAs.mapped.folded.forPrediction");

                System.out.println("\nlib"+i+"_"+j);
                adaptfolding(infile,outfile);
            }
        }
    }

    private static void adaptfolding(File infile, File outfile) {
        System.out.println("Adapting...");
        //File temp = new File(infile+".temp");
        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile)));
            String line = br.readLine();
            int compteur=0;
            while (br.ready()){
                if (line.startsWith(">")){
                    String id = line;
                    String rna = line.substring(1);
                    Vector<String> v = new Vector<String>();
                    line = br.readLine();
                    while (line!=null&&!line.startsWith(">")){
                        v.add(line+"\n"+br.readLine());
                        br.readLine();
                        line = br.readLine();
                        // Cas où il y a des sauts de ligne en trop
                        // excepetion à la dernière ligne du fichier
                        try {
                            while (line.isEmpty()) {
                                line = br.readLine();
                            }
                        } catch (Exception ex) {
                        }
                    }
                    //vérifie que le vecteur a plus d'un object, sinon pas la peine de le
parcourir
                    if (v.size()>=2){
                        int cpt = 0;
                        // Parcourt du vecteur, recherche du brin de RNAfold contenant le
smallRNA
                        for (int i = 0; i < v.size(); i++) {
                            if (v.get(i).contains(rna.replaceAll("T", "U"))){
                                //Cas où une séquence aurait été séparée en 3 et que le rna s'y
retrouverait
```

```
                                //ou que le miRNA est retrouvé dans deux précurseurs coupés issus
du meme
                                //théoriquement rare
                                if (cpt>=1){

pw.println(id.split("\t")[0]+"."+cpt+"\t"+rna+"\n"+v.get(i)+"\n");
                                    System.err.println(id.split("\t")[0]+"\t"+cpt);
                                // cas normal
                                } else {
                                    pw.println(id+"\n"+v.get(i)+"\n");
                                }
                                cpt++;
                            }
                        }
                    } else {
                        pw.println(id+"\n"+v.get(0)+"\n");
                    }
                }
                compteur++;
                if (compteur%10000==0) {
                    System.out.print("*");
                    pw.flush();
                }
            }
            br.close();
            pw.close();

//            while (infile.exists()){
//                infile.delete();
//            }
//            temp.renameTo(infile);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

b_mapping

analyseMapping.java

La majorité des scripts sont exécutés sur un LSF (répartiteur de tache sur un cluster de calcul), et les sorties de chacun d'entre eux sont stockés dans des fichiers. La sortie de MAQ contient des informations sur les statistiques de mappage, récupérées par ce script.

```
/*
 * Permet de lire les outputs des jobs de mapping afin de déterminer
 * un pourcentage de mapping pour chaque taille de chaque librairie
 */

package b_mapping;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;

/**
 *
 * @author Mickael
 */
public class analyseMapping {

    public static int tabtot[][] = new int[10][30];
    public static int tabmap[][] = new int[10][30];

    public static void main(String args[]){
        analyseMapping();
    }

      /**
     * Analyse les fichiers et ressort les pourcentages de mapping
```

```
     */
    public static void analyseMapping(){
        //File serFile= new File("H:\\mik\\_version3\\rnaObject_lib1.ser");

        String f = "H:\\mik\\_version3\\i_mapping\\jobs.genome\\";

        for (int i = 1; i <=10; i++) {
            for (int j = 15; j <= 30; j++) {
                File infile= new File(f+"\\lib"+i+"_"+j+".bsuberr");
                analyse(infile, i, j);
            }
        }

        //print totaux
        System.out.println("");
        try {
            File outfile= new File("H:\\mik\\_version3\\i_mapping\\mapping.counts.txt");
            if (outfile.exists()) outfile.delete();
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
            for (int i = 0; i <16; i++) {
                for (int j = 0; j < 10; j++) {
                    pw.print(tabtot[j][i]+"\t");
                    System.out.print(tabtot[j][i]+"\t");
                }
                pw.print("\n");
                System.out.println("");
            }
            System.out.println("\n");
            pw.println("\n");
            for (int i = 0; i <16; i++) {
                for (int j = 0; j < 10; j++) {
                    pw.print(tabmap[j][i]+"\t");
                    System.out.print(tabmap[j][i]+"\t");
                }
                pw.print("\n");
                System.out.println("");
            }
            pw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }


    }

    public static void analyse(File infile, int lib, int taille){
        //System.out.print("**");
        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            String line = "";
            line = br.readLine();
            while (br.ready()){
                if (line.startsWith("[match_data2mapping]")){
                    line = br.readLine();
                    String tmp = line.substring(36);
                    int total = Integer.valueOf(tmp.split(",")[0]);
                    int mapped = Integer.valueOf(tmp.split(",")[2].trim());
                    tabtot[lib-1][taille-15]= total;
                    tabmap[lib-1][taille-15]= mapped;
                }
                line = br.readLine();
            }

        } catch (Exception e) {
            System.err.println(infile);
            e.printStackTrace();
        }
    }

}
```

mappingQuality.java

Afin d'obtenir une idée de la qualité du séquençage, ce script contrôle la qualité de chaque read qui a été mappé.

```
/*
 * Contrôle la qualité du sequencage apres mapping
 * Les quality values doivent être supérieures ou égales à 10 dans les 10 premières bases
 * A été fait à la fin après prédiction et filtrage, aurait pu être effectué avant aussi...
 */
package b_mapping;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;
import tools.tools;

/**
 * d'abord lancer awk '{print $2}' FS="\t" all_lib.filters_analyses.com | perl -ne '$H{$_}++ or
print'> all_lib.filters_analyses.com.uni.reads
 * @author Mickael
 */
public class mappingQuality {
    public static int lib=0;
    public static HashMap<String,Integer> hmreads;
    public static int tab[]=new int[11];
    public static int tabtot[]=new int[11];

    public static void main(String args[]){
        //predits
        String f = "/ibrixfs1/Data/mik/cutadapt/objects/";
        File predictedmirnafile = new File(f + "all_lib.pred.colorNames.txt");
        File outfile = new File(f + "qualityreads.txt");
        if (outfile.exists())outfile.delete();
        addreadsFileToHashmap(predictedmirnafile);

            for (lib = 1; lib <= 10; lib++) {
            try {
                File qualityfile = new File("/ibrixfs1/Data/mik/_color_reads/" + lib +
"/miRNA_Ble_2_20090727_bcSample1_F3_QV.qual");
                checkQualityInPredicted(qualityfile,outfile);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

//        //tous
//        File outfile = new File("H:\\mik\\_version2\\_color_reads\\qualities.txt");
//        if (outfile.exists())outfile.delete();
//        for (int i = 0; i < tabtot.length; i++) tab[i]=0;
//
//        try {
//            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile,
true)));
//            for (lib = 1; lib <= 10; lib++) {
//                for (int i = 0; i < tab.length; i++) tab[i]=0;
//                try {
//                    pw.println("lib"+lib);
//                    File qualityfile = new File("H:\\mik\\_version2\\_color_reads\\" + lib +
"\\miRNA_Ble_2_20090727_bcSample1_F3_QV.qual");
//                    checkQuality(qualityfile);
//                    for (int i = 0; i < tab.length; i++) {
//                        pw.println(i+"\t"+tab[i]);
//                    }
//                    pw.flush();
//                } catch (Exception e) {
//                    e.printStackTrace();
//                }
//            }
//
//            pw.println("All lib");
//            for (int i = 0; i < tabtot.length; i++) {
//                pw.println(i+"\t"+tabtot[i]);
//            }
```

```
//            pw.close();
//        } catch (Exception e) {
//            e.printStackTrace();
//        }
    }

    private static void checkQuality(File qualityfile) {
        System.out.println("CheckQuality in lib"+lib);

        try {
            BufferedReader br = new BufferedReader(new FileReader(qualityfile));

            String line = "";
            while (br.ready()){
                line = br.readLine();
                if (line.startsWith(">")){
                    String read = line.substring(1);
                    String qual[]=br.readLine().split(" ");
                    int underten=0;
                    for (int i = 0; i < 10; i++) {
                        Integer q = Integer.valueOf(qual[i]);
                        if (q<10) underten++;
                    }
                    tab[underten]++;
                    tabtot[underten]++;
                }
            }
            br.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    private static void checkQualityInPredicted(File qualityfile, File outfile) {
        System.out.println("CheckQuality in lib"+lib);
        try {
            BufferedReader br = new BufferedReader(new FileReader(qualityfile));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
            String line = "";
            while (br.ready()){
                line = br.readLine();
                if (line.startsWith(">")){
                    String read = line.substring(1);
                    if (hmreads.containsKey(read)){
                        String qual[]=br.readLine().split(" ");
                        int underten=0;
                        for (int i = 0; i < 10; i++) {
                            Integer q = Integer.valueOf(qual[i]);
                            if (q<10) underten++;
                        }
                        pw.println(read.replace("_F3","/1")+"\t"+underten+"\t"+lib);
                    }
                }
            }
            br.close();
            pw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    /**
     * Adding reads to hashmap
     * @param predictedmirnafile
     */
    private static void addreadsFileToHashmap(File predictedmirnafile) {
        int countlines = tools.countLines(predictedmirnafile);
        System.out.println("Adding readsFile to hashmap");
        hmreads = new HashMap<String, Integer>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(predictedmirnafile));
            while (br.ready()){
                i++;
```

```
                hmreads.put(br.readLine().replace("/1", "_F3"),1);
            }
            br.close();
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in readsFileToHashmap at "+i+" on "+countlines);
        }
    }


}
```

d_targetgenes

i_parse_tapir.java

Ce script lit les fichiers de sortie de TAPIR afin d'en extraire les résultats sous un format facile à lire. En plus, durant l'étude nous nous sommes procuré des annotations correspondant à certain des gènes inclus dans nos ESTs. Ceci a permis d'avoir quelques nouveaux noms en plus plutôt que des gènes non annotés inconnus.

```
/*
 * Parse les résultats des target genes et va chercher les noms des numeros d'accession
 * dans le fichier genome_ble6DB.fasta et singletons_descriptions.txt
 */

package d_targetgenes;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;

/**
 *
 * @author Mickael
 */
public class i_parse_tapir {
    public static HashMap<String,String> hmtapir = new HashMap<String, String>(10000);
    public static HashMap<String,String> hmgenome;
    //public static HashMap<String,String> hmannot;

    public static void main (String args[]) {
        File genome = new File("/ibrixfs1/Data/mik/genome_ble6DB.clean.fasta");
        addGenomeToHashmap(genome);
        File annotations = new File("/ibrixfs1/Data/mik/singletons_descriptions.txt");
        addAnnotToHashmap(annotations);
        annotations = new File("/ibrixfs1/Data/mik/contigs_annotations.txt");
        addAnnotToHashmap(annotations);

        File tapirFile = new
File("/ibrixfs1/Data/mik/cutadapt/targetgenes/all_lib.mirnas.fasta.tapir");
        File outfile = new
File("/ibrixfs1/Data/mik/cutadapt/targetgenes/tapir_results.summary.txt");
        if (outfile.exists())outfile.delete();
        parseTargetFile(tapirFile, outfile);

    }



    public static void parseTargetFile(File targetFile, File outfile){
        System.out.println("add "+targetFile+" To DB...(* each 10000)");
        String miRNA="";
        String target_acc="";
        double score=0.0;
        //double mfe_ratio=0.00;
        int start=0;
        // seed_gap=0;
        //int seed_mismatch=0;
```

```
//int seed_gu=0;
//int gap=0;
//int mismatch=0;
//int gu=0;
//String miRNA_3="";
//String aln="";
//String target_5="";
int count=0;

try {
    BufferedReader br=new BufferedReader(new FileReader(targetFile));


    String line="";
    while(br.ready()) {
        line=br.readLine().trim();
        if (line.startsWith("miRNA")) {
            count++;
            if (count%10000==0) {
                System.out.print("*");
            }
            miRNA=line.substring(14).replace(">", "");
            line=br.readLine().trim();
            target_acc=line.substring(14);
            line=br.readLine().trim();
            score=Double.valueOf(line.substring(14));
            line=br.readLine().trim();
            //mfe_ratio=Double.valueOf(line.substring(14));
            line=br.readLine().trim();
            start=Integer.valueOf(line.substring(14));
            line=br.readLine().trim();
            //seed_gap=Integer.valueOf(line.substring(14));
            line=br.readLine().trim();
            //seed_mismatch=Integer.valueOf(line.substring(14));
            line=br.readLine().trim();
            //seed_gu=Integer.valueOf(line.substring(14));
            line=br.readLine().trim();
            //gap=Integer.valueOf(line.substring(14));
            line=br.readLine().trim();
            //mismatch=Integer.valueOf(line.substring(14));
            line=br.readLine().trim();
            //gu=Integer.valueOf(line.substring(14));
            line=br.readLine().trim();
            //miRNA_3=line.substring(14);
            line=br.readLine().trim();
            //aln=line.substring(14);
            line=br.readLine().trim();
            //target_5=line.substring(14);
            line=br.readLine().trim();
            try {
                String name = hmgenome.get(target_acc);
                if (name==null){
                    name = "Unknown";
                }
                name = name.trim();
                if (score<=3.0) {
                    String tosave = target_acc + "," + name + "," + start + "," + score;
                    if (hmtapir.get(miRNA) == null) {
                        hmtapir.put(miRNA, tosave);
                    } else {
                        String tmp = hmtapir.get(miRNA);
                        hmtapir.put(miRNA, tmp + ";" + tosave);
                    }
                }
            }catch(Exception e){
                e.printStackTrace();

            }
        }
    }
    br.close();

    System.out.println("Finished");
    System.out.println("number of target genes "+count);

    System.out.println("print Hashmap");
```

```
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
            int i=0;
            for (String s : hmtapir.keySet()) {
                pw.println(s+"\t"+hmtapir.get(s));
                if (count%10000==0) {
                    System.out.print("*");
                }
                i++;
            }
            pw.flush();
            pw.close();
        } catch(Exception e){
            e.printStackTrace();
        }
    }

     /**
     * Add repeat masker results
     * @param genome
     */
    private static void addGenomeToHashmap(File genome) {

        //int countlines = tools.tools.check_number_contigs(genome);
        int countlines = 1756788;
        System.out.println("Adding genome to hashmap");
        int i=0;
        hmgenome = new HashMap<String,String>(countlines+50000,10000);
        try {
            BufferedReader br = new BufferedReader(new FileReader(genome));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine().trim();
                if (line.startsWith(">")) {
                    String acc = null;
                    String name = null;
                    line = line.replace("\t", " ").replace(",", " ");
                    if (line.contains(" ")) {
                        acc = line.substring(1, line.indexOf(" "));
                        name = line.substring(line.indexOf(" ") + 1).replaceAll(",", "
").replaceAll(";", " ");
                        hmgenome.put(acc, name);
                    } else {
                        acc=line.substring(1);
                        hmgenome.put(acc,"unknown");
                    }
                }
            }
            System.out.println("genome hashmap size : "+hmgenome.size()+" on "+countlines);
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in addgenomeHashmap at "+i+" on "+countlines);
        }
    }

    private static void addAnnotToHashmap(File annotations) {

        //int countlines = tools.tools.countLines(annotations);
        int countlines = 11247;
        int detected=0;
        System.out.println("Adding annotations to hashmap");
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(annotations));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                String acc = line.split("\t")[0].trim();
                String name = line.split("\t")[2].trim().replaceAll(",", " ").replaceAll(";", "
");

                if (hmgenome.get(acc)!=null) {
                    detected++;
                }
                hmgenome.put(acc, name);
```

```
            )
            System.out.println("genome hashmap size : "+hmgenome.size()+" with "+countlines+".
Detected : "+detected);
        ) catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in addgenomeHashmap at "+i+" on "+countlines);
        )
    }

)
```

ii_geneOntology.java

Certaines annotations des ESTs contiennent des identifications de référence à des bases de données protéiques (uniprot). Extraire ces références est utile pour ensuite aller chercher leurs fonctions sur gene Ontology, afin de pouvoir identifier quel processus biologique est ciblé par le microARN.

```
/*
 * nettoie le fichier des target genes, recherche les noms pour les retrouver sur gene ontology
 */
package d_targetgenes;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import tools.targetobject;

/**
 *
 * @author Mickael
 */
public class ii_geneOntology {

    public static void main (String args[]) {
        String f = "/ibrixfs1/Data/mik/cutadapt/targetgenes/";
        File infile = new File(f+"tapir_results.summary.txt");
        File genes = new File(f+"tapir_results_genes.txt");
        extractgenes(infile,genes);
        //searchforgenes(infile,genes);
    )


    private static void extractgenes(File infile, File genes) {
        System.out.println("Extracting...");
        int total=0;
        int gen=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(genes)));
            HashMap <String,String> hm = new HashMap<String, String>();
            String line = "";
            while (br.ready()){
                line = br.readLine();
                String seq = line.split("\t")[0];
                String targets[]=line.split(";");
                for (String target : targets) {
                    total++;
                    String tab[]=target.split(",");
                    String acc = tab[0];
                    acc=acc.replace(seq, "").trim();
                    String contig = tab[1];
                    Double score = Double.valueOf(tab[3]);

if(contig.toLowerCase().contains("uniref100")&&!hm.containsKey(contig)&&score<3.5){
                        gen++;
                        //String geneName=getGeneName(contig);
                        String geneName=getUniProt(contig);
                        //String geneName=getGBK(acc);
```

```
                    //String geneName=getGI(acc);
                    hm.put(geneName, geneName);

                }
            }
        }

        for (String s : hm.keySet()) {
            String gene=hm.get(s);
            if(!gene.equals("")){
                pw.println(hm.get(s));
            }

        }
        pw.close();
        System.out.println("\nTotal target genes="+total);
        System.out.println("Total genes="+hm.size());
    }
    catch(Exception e){
        e.printStackTrace();
    }
}

private static String getGI(String acc) {
    if (acc.contains("|")){
        String tab[]=acc.split("\\|");
        acc=tab[1];
    } else acc="";
    return acc;
}

private static String getGBK(String acc) {
    if (acc.contains("|")){
        String tab[]=acc.split("\\|");
        acc=tab[tab.length-1];
    }
    if(acc.contains("Contig")){
        acc="";
    }

    return acc;
}

private static String getUniProt(String contig) {
    if(contig.contains("UniRef")){
        int idx = contig.indexOf("UniRef")+10;
        contig = contig.substring(idx);
        contig = contig.substring(0,contig.indexOf(" "));
    } else {
        contig="";
    }

    return contig;
}

private static String getGeneName(String contig) {

    if (contig.toLowerCase().contains("cluster:")){
//          int idx=contig.toLowerCase().indexOf("cluster")+8;
//          contig = contig.substring(idx);
//          if(contig.contains("n=")){
//              contig=contig.substring(0, contig.indexOf("n="));
//          }
        contig="";
    }

    if(contig.trim().toLowerCase().startsWith("chromosome")&&
            contig.toLowerCase().contains("scaffold")&&
            contig.toLowerCase().contains("whole genome shotgun sequence")){
        contig="";
    }

    //Cas hypothetical protein
    if(contig.toLowerCase().contains("hypothetical protein")){
        contig="";
    }
```

```
        //Cas [species]
        contig = contig.replaceAll("\\[.*\\]", "");

        //Cas Os08g0398700 protein

if(contig.trim().startsWith("Os")&&contig.trim().endsWith("protein")&&contig.trim().split("
").length==2){
            contig="";
        }

        //Cas OSJNBa0044K18.22 protein

if(contig.trim().startsWith("OS")&&contig.trim().endsWith("protein")&&contig.trim().split("
").length==2){
            contig="";
        }


        //Cas 518   0   518 ABI
        if (contig.replaceAll("[0-9]", "").trim().equals("ABI")){
            contig="";
        }

        //399   0   399 SCF
        if (contig.replaceAll("[0-9]", "").trim().equals("SCF")){
            contig="";
        }

        return contig.trim();
    }


    private static void searchforgenes(File infile, File genes) {
        System.out.println("Searching...");
        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            String line = "";
            ArrayList<targetobject> al;
            while (br.ready()){
                line = br.readLine();
                al= new ArrayList<targetobject>();

                String seq = line.split("\t")[0];
                String targets[] = line.split("\t")[0].split(";");

                for (String target : targets) {
                    targetobject to = new targetobject();
                    to.setSequence(seq);
                    String tab[]=target.split(",");
                    to.setAccession(tab[0]);
                    to.setContigName(tab[1]);
                    to.setPosition(Integer.valueOf(tab[2]));
                    to.setScore(Double.valueOf(tab[3]));
                    to.setGeneName(getGeneName(tab[1]));

                    al.add(to);
                }

            }
            br.close();


        } catch (Exception e) {
        }
    }

}
```

iii_miRNAsTargetGenesStats.java

Recherche les statistiques des gènes cibles, selon ceux qui ont des références protéiques ou non, ceux qui sont inconnus, et ceux identifiables par d'autres noms.

```
/*
```

```
 * Statistics des targets genes
 */
package d_targetgenes;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;

/**
 *
 * @author Mickael
 */
public class v_miRNAsTargetGenesStats {
    public static void main(String args[]){
        String f = "H:\\mik\\_version3\\objets\\afterFilters_analyses\\";
        File infile = new File(f+"all_lib.filters_analyses.com.mirnaTargets.txt");
        File outfile = new File(f+"all_lib.filters_analyses.com.mirnaTargets.tabbed.txt");

        int totalmirnas=0;
        int totaltargets=0;
        int unknown=0;
        int uniref=0;
        int unknownuniref=0;
        int unknownonly=0;
        int unirefonly=0;
        int other=0;
        int max=0;
        HashMap<String,Integer> hm = new HashMap<String,Integer>(111000);
        HashMap<String,Integer> hmprint = new HashMap<String,Integer>(111000);
        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile)));
            String line = "";
            while (br.ready()){
                line = br.readLine();
                totalmirnas++;
                String tabs[]=line.split(";");
                int numberTargets=0;
                String seq = tabs[0];
                for (int i=1;i<tabs.length;i++) {


                    if(tabs[i].trim().length()>0){
                        hmprint.put(seq+"\t"+tabs[i].replace(",", "\t").replace("\"",
"").replace(".", ","),0);
                        numberTargets++;
                        totaltargets++;

                        if (tabs[i].toLowerCase().contains("unknown")) {
                        unknown++;
                        }
                        if (tabs[i].toLowerCase().contains("uniref")){
                            uniref++;
                        }
                        String key=tabs[i].split(",")[0];
                        if (hm.containsKey(key)){
                            int tmp=hm.get(key);
                            tmp=tmp+1;
                            hm.put(key, tmp);
                        } else hm.put(key, 1);
                    }
                }
                if
(line.toLowerCase().contains("unknown")&&line.toLowerCase().contains("uniref")){
                    unknownuniref++;
                }
                if
(line.toLowerCase().contains("unknown")&&!line.toLowerCase().contains("uniref")){
                    unknownonly++;
                }
                if
(!line.toLowerCase().contains("unknown")&&line.toLowerCase().contains("uniref")){
```

```
                unirefonly++;
            }
            if
(!line.toLowerCase().contains("unknown")&&!line.toLowerCase().contains("uniref")){
                other++;
            }
            if (numberTargets>max) max=numberTargets;
        }
        for (String s:hmprint.keySet()) {
            pw.println(s);
        }
        pw.close();
        br.close();
        System.out.println("Total miRNAs="+totalmirnas);
        System.out.println("Total targets="+totaltargets);
        System.out.println("Total uniques targets="+hm.size());
        System.out.println("unknown targets="+unknown);
        System.out.println("uniref targets="+uniref);
        System.out.println("other targets="+(totaltargets-uniref-unknown));
        System.out.println("\nmiRNAs having targets unknown AND uniref="+unknownuniref);
        System.out.println("miRNAs having targets unknown ONLY="+unknownonly);
        System.out.println("miRNAs having targets uniref ONLY="+unirefonly);
        System.out.println("miRNAs having targets no unknown AND no uniref="+other);
        System.out.println("\nMaximum targets for one miRNA="+max);
        System.out.println("Mean of target genes by miRNA="+(totaltargets/totalmirnas));

    } catch (Exception e) {
        e.printStackTrace();
    }
  }

}
```

e_filters

i_parseRibosomaux.java

Des blasts furent effectués sur ncbi contre toutes leurs bases de données génomiques pour chercher si nos microARNs ressemblaient à des ribosomaux. Les sorties de ncbi sont en format xml, et ce script permet d'extraire les informations nécessaires.

```
/*
 *  Parse les fichiers de blast et cherche les ribosomaux
 */

package e_filters;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;

/**
 *
 * @author Mickael
 */
public class i_parseRibosomaux {

    public static HashMap<String,String> hmblast = new HashMap<String,String>();
    public static String blastFolder="/ibrixfs1/Data/mik/cutadapt/blasts/ribosomaux/";

    public static void main(String args[]){
        File concatenedxmls = new File(blastFolder+"all_lib.pred.analyses.com.uni.mirnas.xml");
        addRibosomesToHashmapConcat(concatenedxmls);

        File outfile = new File(blastFolder+"all_lib.pred.analyses.com.uni.mirnas.ribosomals");
        printRibosomaux(outfile);

    }
     /**
     * Lit les fichiers xml de résultats de blast et ajoute les miRNAs qui sont
     * détectés par le blast
```

```
 *
 */
private static void addRibosomesToHashmapConcat(File concatenedxmls) {
    System.out.println("Parsing blast files");
    int cpt=0;
    hmblast = new HashMap<String,String>();
    try {
        System.out.println(concatenedxmls.toString());
        BufferedReader br = new BufferedReader(new FileReader(concatenedxmls));
        String line="";
        while (br.ready()){
            cpt++;
            line = br.readLine();
            if (line.startsWith("<Iteration>")){
                cpt++;
                boolean ribo=false;
                br.readLine();br.readLine();
                line=br.readLine();
                String miRNA= line.substring(23, line.indexOf("</"));
                while(!line.startsWith("</Iteration>")&&ribo==false){
                    line = br.readLine();
                    cpt++;
                    if (line.startsWith("  <Hit_def>")){
                        if (line.toLowerCase().contains("ribosomal")){
                            ribo=true;
                            hmblast.put(miRNA,line.substring(11, line.indexOf("</")));
                        }
                    }
                }
            }
        }
        br.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }

}

/**
 * Lit les fichiers xml de résultats de blast et ajoute les miRNAs qui sont
 * détectés par le blast
 *
 */
private static void addRibosomesToHashmap() {
    System.out.println("Parsing blast files");
    int cpt=0;
    hmblast = new HashMap<String,String>();
    for (int i = 0; i <=6; i++) {

        try {
            File INfastafile = new File(blastFolder+i+"000_to_"+i+"999.fasta.xml");
            System.out.println(INfastafile.toString());
            BufferedReader br = new BufferedReader(new FileReader(INfastafile));
            String line="";
            while (br.ready()){
                cpt++;
                line = br.readLine();
                if (line.startsWith("<Iteration>")){
                    cpt++;
                    boolean ribo=false;
                    br.readLine();br.readLine();
                    line=br.readLine();
                    String miRNA= line.substring(23, line.indexOf("</"));
                    while(!line.startsWith("</Iteration>")&&ribo==false){
                        line = br.readLine();
                        cpt++;
                        if (line.startsWith("  <Hit_def>")){
                            if (line.toLowerCase().contains("ribosomal")){
                                ribo=true;
                                hmblast.put(miRNA,line.substring(11, line.indexOf("</")));
                            }
                        }
                    }
                }
            }
            br.close();
```

```
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }


    /**
     * Affiche le contenu du hashmap
     *
     */
    public static void printRibosomaux(File outfile){
        try {
            int detected=0;
            if (outfile.exists()) outfile.delete();
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
            for (String s : hmblast.keySet()) {
                pw.println(s+"\t"+hmblast.get(s));
                System.out.println(s+"\t"+hmblast.get(s));
                detected++;
            }
            pw.close();
            System.out.println("Detected : "+detected);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


}
```

ii_parseLowComplexityRM.java

Afin de faciliter l'organisation des données, les fichiers en FASTA sous forme de >nom\nSequence ont souvent été écrit sous la forme >Sequence\nSequence. Or, certains logiciels comme Repeat Masker refusent des séquences à la place de nom. Les séquences furent donc numérotées, traitées par Repeat Masker et ce script a refait la correspondance des noms et numéros.

```
/*
 *
 * Since RepeatMasker doesn't support sequence in fasta names, had to replace them by numbers
 * with this command : cat all_lib.mirnas.txt | awk  '{print ">predMirna_i\n"$1}' | awk -F_
'/i/{$NF=i++}1' i=0 OFS=_ > all_lib.mirnas.num.fasta
 * Also had to replace all U by T
 * Run RepeatMasker before,
 * This script put back sequences instead of numbers
 */
package e_filters;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;

/**
 *
 * @author Mickael
 */
public class ii_parseLowComplexityRM {

    public static void main(String args[]){
        File RMfile= new
File("/ibrixfs1/Data/mik/cutadapt/repeatMasker_results/all_lib.mirnas.numT.fasta.cat");
        File mirnaFile=new
File("/ibrixfs1/Data/mik/cutadapt/repeatMasker_results/all_lib.mirnas.numT.fasta");
        File outfile= new
File("/ibrixfs1/Data/mik/cutadapt/repeatMasker_results/all_lib.mirnas.RepeatMasker.txt");
        System.out.println("Start...");

        try {
```

```
            BufferedReader brmirnas = new BufferedReader(new FileReader(mirnaFile));
            HashMap<String,String> hm = new HashMap<String, String>();
            String line="";
            while (brmirnas.ready()){
                line = brmirnas.readLine();
                if (line.startsWith(">")){
                    String name = line.substring(1);
                    String mirna = brmirnas.readLine();
                    hm.put(name, mirna);
                }
            }
            brmirnas.close();

            BufferedReader brRM = new BufferedReader(new FileReader(RMfile));
            PrintWriter pw = new PrintWriter(new FileWriter(outfile));
            while (brRM.ready()){
                line = brRM.readLine();
                try {
                    String l[] = line.split(" ");
                    String name = l[4];
                    String mirna = hm.get(name).replace("T", "U");

                    pw.println(line.replace(name, mirna));
                } catch (Exception e) {
                    //pw.println(line);
                }
            }
            brRM.close();
            pw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("Finished...");
    }

}
```

f_analyses

checkBlasts.java

Parcourt les fichiers de blast et accepte certains hits comme vrais ou faux selon nos paramètres dans matériel et méthodes.

```
/*
 * Check blast against mirBase/PMRD or non codants to get conserved.
 * Blast is considered as true depending coverage, mismatchs and gaps
 */
package f_analyses;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;
import tools.tools;


/**
 *
 * @author Mickael
 */
public class checkBlasts {
    public static HashMap<String,String> hmmirna= new HashMap<String,String>();

    public static void main(String args[]){

        //CheckAgainstmiRNAsDB();
        CheckAgainstncRNAs();
    }


    private static void CheckAgainstncRNAs() {
```

```
        File blastFile= new
File("/ibrixfs1/Data/mik/cutadapt/blasts/all_lib.mirnas.blastncrna.txt");
        File trueblastFile = new
File("/ibrixfs1/Data/mik/cutadapt/blasts/all_lib.mirnas.blastncrna.true.txt");

        int tru=0;
        int cpt=0;
        System.out.println("Reading blastFile");

        try {
            BufferedReader br = new BufferedReader(new FileReader(blastFile));
            PrintWriter pw = new PrintWriter(new FileWriter(trueblastFile));
            pw.println("Query sequence\tQuery Length\tSubject name"
                    + "\tStart of alignment in query\tEnd of alignment in query"
                    + "\tStart of alignment in subject\tEnd of alignment in subject"
                    + "\tAligned part of query sequence\tAligned part of subject sequence"
                    + "\tAlignment length\tPercentage of identical matches"
                    + "\tNumber of identical matches\tNumber of mismatches"
                    + "\tInternal gaps\tExpect value\tStrand\tQuery coverage");

            String line = "";
            while (br.ready()){
                cpt++;
                line = br.readLine();
                String infos[] = line.split(",");
                String qseqid = infos[0];
                String sallseqid = infos[1];
                int qstart = Integer.valueOf(infos[2])-1;
                int qend = Integer.valueOf(infos[3]);
                int sstart = Integer.valueOf(infos[4])-1;
                int send = Integer.valueOf(infos[5]);
                int length = Integer.valueOf(infos[8]);
                double pident = Double.valueOf(infos[9]);
                int gaps = Integer.valueOf(infos[12]);
                int mismatchs = Integer.valueOf(infos[11]);

                 //Calculate strand
                String strand;
                if (sstart<send){
                    strand = "+";
                } else strand = "-";

                //Determine conserved
                boolean conserved=false;

                //Determine external gaps
                //Calculate qcoverage (query coverage)
                double qcoverage=0.0;

                //Calculate qcoverage (query coverage)
                qcoverage = ((double)qend-(double)qstart)/(double)qseqid.length();

                //for lengths 15,16
                if (qseqid.length()<=16){
                    if (pident>=80&&qcoverage>=1.0&&gaps<3){
                    conserved=true;
                    } else conserved=false;

                    //others lengths
                }else if (pident>=80&&qcoverage>=0.9&&gaps<3){
                    conserved=true;
                } else conserved=false;

                //check if not miRNA in ncrna
                if (sallseqid.contains("miRNA")){
                    conserved=false;
                }

                //print true blasts
                if (conserved) {
                    pw.println(qseqid+"\t"+qseqid.length()
                            +line.replaceAll(",", "\t").substring(line.indexOf(","))
                            +"\t"+strand+"\t"+qcoverage);
                    tru++;
                }
```

```java
            }
            br.close();
            pw.close();

        } catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("true = "+tru+" on "+cpt+" hits");
    }

    public static void CheckAgainstmiRNAsDB(){
//        File mirnas=new
File("/ibrixfs1/Data/mik/cutadapt/blasts/databases/miRbase.mirnasT.fasta");
//        File blastFile= new
File("/ibrixfs1/Data/mik/cutadapt/blasts/all_lib.smallrnas.blastmirbase.txt");
//        File trueblastFile = new
File("/ibrixfs1/Data/mik/cutadapt/blasts/all_lib.smallrnas.blastmirbase.true.txt");

        File mirnas=new File("/ibrixfs1/Data/mik/cutadapt/blasts/databases/PMRD.mirnasT.fasta");
        File blastFile= new
File("/ibrixfs1/Data/mik/cutadapt/blasts/all_lib.smallrnas.blastPMRD.txt");
        File trueblastFile = new
File("/ibrixfs1/Data/mik/cutadapt/blasts/all_lib.smallrnas.blastPMRD.true.txt");

//        String f = "H:/mik/_version3/viii_analyses/Patrick/";
//        File mirnas=new File(f+"smallRNAs_matching_to_genome_MAQ.fasta");
//        File blastFile= new File(f+"mirbase_mature_count.blastsmallRNA");
//        File trueblastFile = new File(f+"mirbase_mature_count.blastsmallRNA.true.txt");

        int tru=0;
        int cpt=0;
        addmirnastoHM(mirnas);
        System.out.println("Reading blastFile");
        try {
            BufferedReader br = new BufferedReader(new FileReader(blastFile));
            PrintWriter pw = new PrintWriter(new FileWriter(trueblastFile));
            pw.println("Query sequence\tQuery Length\tSubject sequence\tSubject Length\tSubject
name"
                    + "\tStart of alignment in query\tEnd of alignment in query"
                    + "\tStart of alignment in subject\tEnd of alignment in subject"
                    + "\tAligned part of query sequence\tAligned part of subject sequence"
                    + "\tAlignment length\tPercentage of identical matches"
                    + "\tNumber of identical matches\tNumber of mismatches"
                    + "\tInternal gaps\tExpect value\tStrand\tQuery coverage"
                    + "\tSubject coverage\tExternal gaps");

            String line = "";
            while (br.ready()){
                cpt++;
                line = br.readLine();
                String infos[] = line.split(",");
                String qseqid = infos[0];
                String sallseqid = infos[1];
                int qstart = Integer.valueOf(infos[2])-1;
                int qend = Integer.valueOf(infos[3]);
                int sstart = Integer.valueOf(infos[4])-1;
                int send = Integer.valueOf(infos[5]);
                int length = Integer.valueOf(infos[8]);
                double pident = Double.valueOf(infos[9]);
                int gaps = Integer.valueOf(infos[12]);
                int mismatchs = Integer.valueOf(infos[11]);

                //get sseqid from fasta files
                String sseqid=hmmirna.get(sallseqid);

                if (sseqid==null){
                    //System.err.println("Error at "+sallseqid);
                    sseqid="NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN";
                }
                sseqid= sseqid.replaceAll("U", "T");

                 //Calculate strand
                String strand;
                if (sstart<send){
                    strand = "+";
```

```
                } else strand = "-";

                //Determine conserved
                boolean conserved=false;

                //Determine external gaps
                //int externalGaps=Math.abs(qseqid.length()-sseqid.length());
                //Calculate qcoverage (query coverage)
                double qcoverage=0.0;
                double scoverage=0.0;

                //calculate scoverage (subject coverage)
                scoverage = ((double)send-(double)sstart)/(double)sseqid.length();

                //Calculate qcoverage (query coverage)
                qcoverage = ((double)qend-(double)qstart)/(double)qseqid.length();

                //for lengths 15,16
                if (qseqid.length()<=16){
                    if (pident>=80&&qcoverage>=1.0&&scoverage>=0.9&&gaps<3){
                    conserved=true;
                    } else conserved=false;

                    //others lengths
                }else if (pident>=80&&qcoverage>=0.9&&scoverage>=0.9&&gaps<3){
                    conserved=true;
                } else conserved=false;

                //External gaps
                int extgaps=0;
                extgaps=Math.abs((qseqid.length()-(qend-qstart)))+Math.abs(sseqid.length()-(send-
sstart));


                //print true blasts
                if (conserved) {
                    pw.println(qseqid+"\t"+qseqid.length()+"\t"
                            +sseqid+"\t"+sseqid.length()
                            +line.replaceAll(",", "\t").substring(line.indexOf(","))
                            +"\t"+strand+"\t"+qcoverage+"\t"+scoverage+"\t"+extgaps);
                    tru++;
                }

            }
            br.close();
            pw.close();


        } catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("true = "+tru+" on "+cpt+" hits");
    }

    private static void addmirnastoHM(File mirnas) {
        int countlines = tools.check_number_contigs(mirnas);
        System.out.println("Adding mirnas to hashmap");
        hmmirna = new HashMap<String, String>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(mirnas));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                if (line.startsWith(">")){
                    String tab[]=line.split(" ");
                    String name = tab[0].substring(1);
                    String seq = br.readLine();
                    hmmirna.put(name, seq);
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in mirnaPrecFileToHashmap at "+i+" on "+countlines);
```

```
        )
    )

)
```

checkRNAComplements.java

Vérifie la présence du reverse complément ou complément des données mappées ou celles après prédiction dans les fichiers FASTA de sortie de cutadapt. Ce test est effectué afin de vérifier leur présence dans le séquençage.

```
/*
 * Prend un objet contenant les données originales de mapping ou celles des prédictions
 * Récupère les miRNAs originaux après mapping, prend leur reverse complement et leur complement
 * Vérifie la présence du reverse complément ou complément dans les fichiers fasta de sortie de
cutadapt
 *
 */

package f_analyses;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.ObjectInputStream;
import java.io.PrintWriter;
import java.util.HashMap;
import tools.rnaobject;
import tools.tools;

/**
 *
 * @author Mickael
 */
public class checkRNAComplements {

    public static String objectsFolder="H:\\mik\\_version3\\objets\\";
    public static String fastaFolder="H:\\mik\\_version3\\i_mapping\\cutadapt\\fasta\\";
    public static HashMap<String,String> hmfasta ;

    public static void main(String args[]){
        checkComplement();
    }

    public static void checkComplement(){

        for (int i = 1; i <=10; i++) {
            for (int j = 15; j <= 30; j++) {
//                File INobject= new File(objectsFolder+i+"\\lib"+i+"_"+j+".obj");
//                File INfasta = new File(fastaFolder+i+"\\lib"+i+"_"+j+".fasta");
//                File outfile = new
File(objectsFolder+"\\Count.complements.fromMapping.obj.txt");
                File INobject= new File(objectsFolder+i+"\\lib"+i+"_"+j+".pred.obj");
                File INfasta = new File(fastaFolder+i+"\\lib"+i+"_"+j+".fasta");
                File outfile = new
File(objectsFolder+"\\Count.complements.fromPrediction.obj.txt");

                if (outfile.exists()) outfile.delete();

                int comp=0;
                int revcomp=0;
                addfastaToHashmap(INfasta);
                rnaobject r=null;
                try {
                    int cpt=0;
                    PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile,
true)));
                    System.out.print("lib"+i+"_"+j);
                    pw.print("lib"+i+"_"+j);
                    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(INobject));
```

```
                boolean b=true;
                while (b) {
                    r = (rnaobject) ois.readObject();
                    //teste fin du fichier pour éviter une exception
                    if (r.getId()==0){
                        b = false;
                    } else {
                        String rna = r.getOriginalSmallRNAsequence();
                        String complement = tools.dnaComplement(rna);
                        String reversecomplement = tools.dnaReverseComplement(rna);
                        if (hmfasta.get(reversecomplement)!=null){
                            revcomp++;
                        }
                        if (hmfasta.get(complement)!=null){
                            comp++;
                        }
                    }
                    cpt++;
                    pw.flush();
                }
                System.out.print("\t"+revcomp);
                pw.print("\t"+revcomp);
                System.out.print("\t"+comp);
                pw.print("\t"+comp);
                System.out.println("\t"+cpt);
                pw.println("\t"+cpt);
                ois.close();
                pw.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        )
    }
}

private static void addfastaToHashmap(File INfastafile) {
    int countlines = tools.check_number_contigs(INfastafile);
    int i=0;
    hmfasta = new HashMap<String,String>(countlines+100,10000);
    try {
        BufferedReader br = new BufferedReader(new FileReader(INfastafile));
        String line="";
        while (br.ready()){
            i++;
            line = br.readLine();
            if (line.startsWith(">")){
                hmfasta.put(br.readLine(),line.substring(1));
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        System.err.println("Error in addExprHashmap at "+i+" on "+countlines);
    }
}
}
```

conservedAgainstMirnasDBsItol.java

Ce script ressemble à celui de checkblast.java, mais ne fait pas que vérifier le contenu des blasts. Selon les microARNs conservés du blast, il créé les fichiers acceptés par Itol afin de construire par la suite des arbres phylogénétiques.

```
/*
 * Extract statistics from mirbase and PMRD from blast
 * Prepare for itol
 * output :
 * sigle+"\t"+name+"\t"+count
 *
 */

package f_analyses;

import java.io.BufferedReader;
```

```
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;

/**
 *
 * @author Mickael
 */
public class conservedAgainstMirnasDBsItol {

    public static File mirbaseStats;
    public static File PMRDStats;

    /**
     * Hashmaps of databases lists
     * HashMap<Sigle, CompleteName>
     */
    public static HashMap<String, String> hmmirbaselist = new HashMap<String, String>();
    public static HashMap<String, String> hmpmrdlist = new HashMap<String, String>();

    /**
     * Hashmap of fasta content
     * HashMap<miRNAname, Sequence>
     */
    public static HashMap<String, String> hmmirbasefasta = new HashMap<String, String>();
    public static HashMap<String, String> hmpmrdfasta = new HashMap<String, String>();

    /**
     * Hashmap of databases content
     * count is the number of miRNA for each species in databases
     * HashMap<Sigle, count>
     */
    public static HashMap<String, Integer> hmmirbaseCount = new HashMap<String, Integer>();
    public static HashMap<String, Integer> hmpmrdCount = new HashMap<String, Integer>();

    /**
     * Hashmap of species ncbi taxonomy ids
     * ids can be int or names, depending if they are recognized by itol
     * HashMap<Sigle, String>
     */
    public static HashMap<String, String> hmmirbasetax = new HashMap<String, String>();
    public static HashMap<String, String> hmpmrdtax = new HashMap<String, String>();



    /**
     *
     * @param args
     */
    public static void main(String args[]){
        System.out.println("Step 1 : Getting names and sequences from fasta files");
        parseFastafiles();
        System.out.println("Step 2 : Getting sigles and full names from html files");
        parseHTMLfiles();
        System.out.println("Step 3 : Getting full names and Taxonomy");
        parseTaxonomyfiles();
        System.out.println("Step 4 : Check conservation of smallRNAs and create itol files");
        smallRNAsConserved();
        System.out.println("Step 5 : Check conservation of miRNAs and create itol files");
        miRNAsConserved();
        System.out.println("Step 6 : Adapt to itol");
        miRNAsConserved();
    }

   private static void parseTaxonomyfiles() {
       String f = "/ibrixfs1/Data/mik/cutadapt/blasts/itol/";

        File mirbasetax = new File(f+"mirbase.IDs.compatible.itol.txt");
        File pmrdtax = new File(f+"PMRD.IDs.compatible.itol.txt");

        System.out.println("Parsing mirbase taxonomy");
        parseTaxonomy(mirbasetax,hmmirbasetax);
        System.out.println("Parsing pmrd taxonomy");
```

```
        parseTaxonomy(pmrdtax,hmpmrdtax);
    }

    private static void parseFastafiles() {
        String f = "/ibrixfs1/Data/mik/cutadapt/blasts/itol/";

        File mirbasefasta = new File(f+"mirbase.mature.fasta");
        File pmrdfasta = new File(f+"pmrd.mature.fasta");

        File outmirbasefasta = new File(f+"mirbaseStatsFromfasta.txt");
        File outpmrdfasta = new File(f+"pmrdStatsFromfasta.txt");

        System.out.println("Parsing mirbase fasta");
        parsefasta(mirbasefasta,hmmirbasefasta,hmmirbaseCount,outmirbasefasta);
        System.out.println("Parsing pmrd fasta");
        parsefasta(pmrdfasta,hmpmrdfasta,hmpmrdCount,outpmrdfasta);
    }


    private static void parsefasta(File fasta, HashMap<String, String> hmfasta,
            HashMap<String, Integer>  hmcount,File StatsFile) {

        try {
            BufferedReader br = new BufferedReader(new FileReader(fasta));

            String line = "";
            while (br.ready()){
                line = br.readLine();
                if (line.startsWith(">")){
                    String name = null;
                    try {
                        name = line.substring(1, line.indexOf(" "));
                    } catch (Exception e) {
                        name = line.substring(1);
                    }
                    String seq = br.readLine();
                    hmfasta.put(name, seq);
                    String sigle = name.substring(0, name.indexOf("-"));
                    if (hmcount.containsKey(sigle)){
                        int tmp = hmcount.get(sigle);
                        tmp++;
                        hmcount.put(sigle, tmp);
                    } else hmcount.put(sigle, 1);
                }
            }
            br.close();

            //print sigles and occurences for each species
            PrintWriter pw = new PrintWriter(new FileWriter(StatsFile));
            for (String s : hmcount.keySet()) {
                pw.println(s+"\t"+hmcount.get(s));
            }
            pw.close();

        } catch (Exception e){
            e.printStackTrace();
        }
    }


    /**
     *
     */
    private static void miRNAsConserved() {
        String f = "/ibrixfs1/Data/mik/cutadapt/blasts/";
        //mirbase
        File miRNAsVSmirbase = new File (f+"all_lib.mirnas.blastmirbase.txt");
        File outmirbase = new File (f+"all_lib.mirnas.blastmirbase.true.txt");
        File outmirbasecount = new File (f+"all_lib.mirnas.blastmirbase.count.txt");
        File outmirbaseitol = new File (f+"all_lib.mirnas.blastmirbase.itol.txt");
        System.out.println("parsing miRNAs vs mirbase");
        parsevsDB(miRNAsVSmirbase, outmirbase, outmirbasecount,outmirbaseitol);

        //pmrd
        File miRNAsVSpmrd = new File (f+"all_lib.mirnas.blastPMRD.txt");
        File outpmrd = new File (f+"all_lib.mirnas.blastPMRD.true.txt");
```

```
        File outpmrdcount = new File (f+"all_lib.mirnas.blastPMRD.count.txt");
        File outpmrditol = new File (f+"all_lib.mirnas.blastPMRD.itol.txt");
        System.out.println("parsing miRNAs vs PMRD");
        parsevsDB(miRNAsVSpmrd, outpmrd,outpmrdcount,outpmrditol);
    }


    /**
     * check for blast results of smallRNAs (after cutadapt)
     */
    private static void smallRNAsConserved() {
        String f = "/ibrixfsl/Data/mik/cutadapt/blasts/";
        //mirbase
        File smallRNAsVSmirbase = new File (f+"all_lib.smallrnas.blastmirbase.txt");
        File outmirbase = new File (f+"all_lib.smallrnas.blastmirbase.true.txt");
        File outmirbasecount = new File (f+"all_lib.smallrnas.blastmirbase.count.txt");
        File outmirbaseitol = new File (f+"all_lib.smallrnas.blastmirbase.itol.txt");
        System.out.println("parsing smallRNAs vs mirbase");
        parsevsDB(smallRNAsVSmirbase, outmirbase, outmirbasecount,outmirbaseitol);

        //pmrd
        File smallRNAsVSpmrd = new File (f+"all_lib.smallrnas.blastPMRD.txt");
        File outpmrd = new File (f+"all_lib.smallrnas.blastPMRD.true.txt");
        File outpmrdcount = new File (f+"all_lib.smallrnas.blastPMRD.count.txt");
        File outpmrditol = new File (f+"all_lib.smallrnas.blastPMRD.itol.txt");
        System.out.println("parsing smallRNAs vs PMRD");
        parsevsDB(smallRNAsVSpmrd, outpmrd,outpmrdcount,outpmrditol);
    }

    /**
     * parse blast results
     * @param blastResult
     * @param outfile
     */
    private static void parsevsDB(File blastResult, File outfile, File count, File itol) {


        //Hashmap qui va contenir les noms des hits
        HashMap<String,Integer> hmCompleteNameCounts= new HashMap<String, Integer>(5000);

        try {
            // lecture du fichier de résultat de blast
            BufferedReader br = new BufferedReader(new FileReader(blastResult));
            // printer des blasts considérés comme true
            PrintWriter pw = new PrintWriter(new FileWriter(outfile));
            pw.println("Query sequence\tQuery Length\tSubject sequence\tSubject Length\tSubject
name"
                    + "\tStart of alignment in query\tEnd of alignment in query"
                    + "\tStart of alignment in subject\tEnd of alignment in subject"
                    + "\tAligned part of query sequence\tAligned part of subject sequence"
                    + "\tAlignment length\tPercentage of identical matches"
                    + "\tNumber of identical matches\tNumber of mismatches"
                    + "\tInternal gaps\tExpect value\tStrand\tQuery coverage"
                    + "\tSubject coverage\tExternal gaps");

            String line = "";
            while (br.ready()){
                line = br.readLine();
                String infos[] = line.split(",");
                String qseqid = infos[0];
                String sallseqid = infos[1];
                int qstart = Integer.valueOf(infos[2])-1;
                int qend = Integer.valueOf(infos[3]);
                int sstart = Integer.valueOf(infos[4])-1;
                int send = Integer.valueOf(infos[5]);
                int length = Integer.valueOf(infos[8]);
                int gaps = Integer.valueOf(infos[12]);
                double pident = Double.valueOf(infos[9]);

                //get sseqid from fasta files
                String sseqid="";
                if (count.toString().toLowerCase().contains("mirbase")){
                    sseqid = hmmirbasefasta.get(sallseqid);
                } else sseqid = hmpmrdfasta.get(sallseqid);
                if (sseqid==null){
                    //System.err.println("Error at "+sallseqid);
```

```
            sseqid="NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN";
        }
        sseqid= sseqid.replaceAll("U", "T");

        //Calculate diff (useless)
        //int diff = qseqid.length()-length;

        //Calculate strand
        String strand;
        if (sstart<send){
            strand = "+";
        } else strand = "-";

        //calculate scoverage (subject coverage)
        double scoverage = ((double)send-(double)sstart)/(double)sseqid.length();

        //Calculate qcoverage (query coverage)
        double qcoverage = ((double)qend-(double)qstart)/(double)qseqid.length();

        //Determine conserved
        boolean conserved;

        //for lengths 15,16
        if (qseqid.length()<=16){
            if (pident>=80&&qcoverage>=1.0&&scoverage>=0.9&&gaps==0){
            conserved=true;
            } else conserved=false;

            //others lengths
        }else if (pident>=80&&qcoverage>=0.9&&scoverage>=0.9&&gaps==0){
            conserved=true;
        } else conserved=false;

        //External gaps
        int extgaps=0;
        extgaps=Math.abs((qseqid.length()-(qend-qstart)))+Math.abs(sseqid.length()-(send-
sstart));


        //print true blasts
        if (conserved) {
            pw.println(qseqid+"\t"+qseqid.length()+"\t"+sseqid+"\t"+sseqid.length()
                    +line.replaceAll(",", "\t").substring(line.indexOf(","))+
                    "\t"+strand+"\t"+qcoverage+"\t"+scoverage+"\t"+extgaps);

            if (hmCompleteNameCounts.containsKey(sallseqid)){
                int tmp = hmCompleteNameCounts.get(sallseqid);
                tmp++;
                hmCompleteNameCounts.put(sallseqid,tmp);
            } else hmCompleteNameCounts.put(sallseqid, 1);
        }

    }
    br.close();
    pw.close();

    //count sigles
    HashMap<String,Integer> hmSiglesCounts= new HashMap<String, Integer>(200);
    boolean goodblast=true;
    for (String completename : hmCompleteNameCounts.keySet()) {
        //System.out.println(completename+"\t"+hmCompleteNameCounts.get(completename));
        String sigle = null;
        try {
            sigle = completename.substring(0, completename.indexOf("-"));
            goodblast=true;
        } catch (Exception e) {
            goodblast=false;
        }

        if (goodblast) {
            if (hmSiglesCounts.containsKey(sigle)) {
                int tmp = hmSiglesCounts.get(sigle);
                tmp++;
                hmSiglesCounts.put(sigle, tmp);
            } else {
                hmSiglesCounts.put(sigle, 1);
```

```
                    )
                )
            )

            //print counts for each sigles
            if (count.toString().toLowerCase().contains("mirbase")){
                printSigles(count, hmSiglesCounts, hmmirbaselist,
hmmirbaseCount,hmmirbasetax,itol);
            } else {
                printSigles(count, hmSiglesCounts, hmpmrdlist, hmpmrdCount,hmpmrdtax,itol);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Print counts for each sigle
     * @param count
     * @param hmSiglesCounts
     * @param hmDBlist
     * @param hmDBcount
     */
    private static void printSigles(File count, HashMap<String,Integer> hmSiglesCounts,
            HashMap<String, String> hmDBlist, HashMap<String, Integer> hmDBcount,
            HashMap<String, String> hmTax, File itol){
        try {
            PrintWriter pwcount = new PrintWriter(new FileWriter(count));
            PrintWriter pwitolcounts = new PrintWriter(new FileWriter(itol));
            //PrintWriter pwitollabels = new PrintWriter(new
FileWriter(itol.toString()+".labels"));

            pwitolcounts.println("LABELS\tCount\tPercentage\nCOLORS\t#ff0000\t#00ff00");

            for (String sigle : hmDBlist.keySet()) {
                String speciesName = hmDBlist.get(sigle).replaceAll(" ", "_");
                int speciesCount = hmDBcount.get(sigle);
                int blastcount = 0;

                try {
                    blastcount = hmSiglesCounts.get(sigle);
                } catch (Exception e) {
                    blastcount = 0;
                }
                if (blastcount > 0) {
                    pwcount.println(sigle + "\t" + speciesName + "\t" + blastcount + "/" +
speciesCount);
                    if (!hmTax.containsKey(speciesName)){
                        System.err.println(speciesName+" not found");
                    } else {
                        //print for itol
                        int perc = blastcount *100/speciesCount;
                        if (blastcount>1) {
                            pwitolcounts.println(hmTax.get(speciesName) + "\t" + blastcount +
"\t" + perc);
                        }

//pwitollabels.println(hmTax.get(speciesName)+"\t"+hmTax.get(speciesName)+"   "+blastcount + "/"
+ speciesCount);
                    }
                }
            }
            pwcount.close();
            pwitolcounts.close();
            //pwitollabels.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

/**
     * Prepare html files
     */
    private static void parseHTMLfiles() {
        String f = "/ibrixfs1/Data/mik/cutadapt/blasts/itol/";
```

```
        File htmlmiRbase= new File(f+"mirbase.htm");
        File htmlPMRD = new File(f+"PMRD.htm");

        mirbaseStats= new File(f+"mirbaseStatsFromhtml.txt");
        PMRDStats= new File(f+"PMRDStatsFromhtml.txt");

        parsepmrd(htmlPMRD,PMRDStats);
        parsemirbase(htmlmiRbase,mirbaseStats);
    }

    /**
     * parse mirbase html file from http://www.mirbase.org/cgi-bin/browse.pl
     * @param htmlmiRbase
     * @param mirbaseStats
     */
    private static void parsemirbase(File htmlmiRbase, File mirbaseStats) {
        System.out.println("parsing html mirbase");
        try {
            BufferedReader br = new BufferedReader(new FileReader(htmlmiRbase));
            PrintWriter pw = new PrintWriter(new FileWriter(mirbaseStats));
            String line = "";
            while (br.ready()){
                line = br.readLine();
                if (line.contains("speciesTag")){
                    //sigle
                    String sigle = "";
                    int start = line.indexOf("org=")+3;
                    while (line.charAt(start+1)!='"'){
                        sigle += line.charAt(start+1);
                        start++;
                    }

                    String name = line.substring(start+3,line.indexOf("</a>"));
                    //String count = line.substring(line.indexOf("(")+1,line.lastIndexOf(")"));
                    pw.println(sigle+"\t"+name);
                    hmmirbaselist.put(sigle,name);
                }
            }
            br.close();
            pw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * parse PMRD html file (from bioinformatics.cau.edu.cn/PMRD/browse.htm)
     * @param htmlPMRD
     * @param PMRDStats
     */
    private static void parsepmrd(File htmlPMRD, File PMRDStats) {
        System.out.println("parsing html PMRD");
        try {
            BufferedReader br = new BufferedReader(new FileReader(htmlPMRD));
            PrintWriter pw = new PrintWriter(new FileWriter(PMRDStats));
            String line = "";
            while (br.ready()){
                line = br.readLine();
                if (line.contains("all_species")){
                    //sigle
                    String sigle = "";
                    int start = line.indexOf("all_species=")+11;
                    while (line.charAt(start+1)!='_'){
                        sigle += line.charAt(start+1);
                        start++;
                    }
                    String name = line.substring(start+37,line.indexOf("</a>")).replace("<span
style=\"color:orange; font-size:18px\">", "").replace("</span>", "");

                    //String count =
line.substring(line.lastIndexOf("(")+1,line.lastIndexOf(")"));
                    pw.println(sigle+"\t"+name);
                    hmpmrdlist.put(sigle,name);
                }
            }
            br.close();
```

```
            pw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void parseTaxonomy(File tax, HashMap<String, String> hmtax) {
        try {
            BufferedReader br = new BufferedReader(new FileReader(tax));
            String line="";
            while (br.ready()){
                line = br.readLine();
                String fullname=line.split("\t")[0];
                String taxo=line.split("\t")[1];

                hmtax.put(fullname, taxo);
            }
            br.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

i_mirnasStars.java

A partir du microARN, du précurseur et de la structure secondaire, il est possible de calculer le microARN* (star), c'est-à-dire son complément.

```
/*
 * Get mirnasStars
 */
package f_analyses;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;

/**
 *
 * @author Mickael
 */
public class i_mirnasStars {
    public static String folderobjects="/ibrixfs1/Data/mik/cutadapt/objects/";

    public static void main(String args[]){
        // obtain infile with :
        //awk '{print $11"\t"$16"\t"$17}' FS="\t" all_lib.pred.analyses.com | perl -ne '$H{$_}++
or print' > forMirnasStars.txt (microARN,precursor,structure)

        File infile = new File(folderobjects+"forMirnasStars.txt");
        File outfile = new File(folderobjects+"mirnasStars.txt");
        if (outfile.exists()) outfile.delete();

        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
            String line="";

            while (br.ready()){
                boolean gap=false;
                line = br.readLine();
                String tab[]=line.split("\t");
                String mirna = tab[0];
                String prec = tab[1];
                String struct = tab[2];
                String star = null;
                if (mirna.contains("-")){
                    gap=true;
                }
                try {
```

```
                if (gap){
                    mirna = mirna.substring(0, mirna.length()-1);
                    star = getStar(mirna, prec, struct);
                    star = star+ "-";
                } else star = getStar(mirna, prec, struct);
            } catch (Exception e) {
                star = "error";
            }
            pw.println(line+"\t"+star+"\t"+star.length());
        }
        br.close();
        pw.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

}


private static String getStar(String mirna, String prec, String struc) {
    if (mirnaInLoop(mirna, prec, struc)) {
        return "loop";
    }

    String star="";
    int taille = mirna.length();
    int mirnaStart=prec.indexOf(mirna);
    int mirnaEnd=mirnaStart+taille;
    int loopStart=struc.lastIndexOf("(");
    int loopEnd=struc.indexOf(")");

    String arm=null;
    try {
        if (struc.substring(mirnaStart, mirnaEnd).contains("(")) {
            arm = "5'";
        } else {
            arm = "3'";
        }
    } catch (Exception e) {
        System.err.println("error at "+mirna);
        return "error";
    }


    if (arm.equals("5'")){

        // Distance du mirna a la loop effecuté en comptant le nombre de parentheses
        int nbrParentheses=0;
        String intervale = null;
        try {
            intervale = struc.substring(mirnaEnd, loopStart);
        } catch (Exception e) {
            intervale="";
        }
        for (char c : intervale.toCharArray()) {
            if (c=='('){
                nbrParentheses++;
            }
        }
        // Determination du départ du mirna star en fonction du nombre de parentheses
        // après la fin de la loop
        int cpt=0;
        int starStart=loopEnd;
        while (cpt!=nbrParentheses){
            if (struc.charAt(starStart)==')'){
                cpt++;
                starStart++;
            } else {
                starStart++;
            }
        }

        //recupération du mirna star
        int posOnStar = starStart;
        boolean dec = false;
```

```
            for (int i = mirnaEnd; i > mirnaStart; i--) {
                char a = struc.charAt(i);
                char b = struc.charAt(posOnStar++);
                if (isParenthese(a)&&isParenthese(b)||a==b){
                    star+=prec.charAt(posOnStar);
                } else if (a=='.'&&isParenthese(b)){
                        i--;
                        posOnStar--;
                        dec = true;
                    } else if (b=='.'&&isParenthese(a)){
                        star+=prec.charAt(posOnStar);
                    }
            }
            if (dec){
                star+=prec.charAt(posOnStar-1);
            }
        }
        ///////////////3'
        else {

            // Distance du mirna a la loop effecuté en comptant le nombre de parentheses
            int nbrParentheses=0;
            String intervale = null;
            try {
                intervale = struc.substring(loopEnd, mirnaStart);
            } catch (Exception e) {
                intervale="";
            }
            for (char c : intervale.toCharArray()) {
                if (c==')'){
                    nbrParentheses++;
                }
            }

            // Determination du départ du mirna star en fonction du nombre de parentheses
            // avant le début de la loop
            int cpt=0;
            int starEnd=loopStart;
            while (cpt!=nbrParentheses){
                if (struc.charAt(starEnd)=='('){
                    cpt++;
                    starEnd--;
                } else {
                    starEnd--;
                }
            }

            //recupération du mirna star
            int posOnStar = starEnd;
            boolean dec = false;
            for (int i = mirnaStart; i < mirnaEnd; i++) {
                if (posOnStar>=0) {
                    char a = struc.charAt(i);
                    char b = struc.charAt(posOnStar);
                    if (isParenthese(a) && isParenthese(b) || a == b) {
                        star = prec.charAt(posOnStar) + star;
                        posOnStar--;
                    } else if (a == '.' && isParenthese(b)) {
                        i++;
                        dec = true;
                    } else if (b == '.' && isParenthese(a)) {
                        star = prec.charAt(posOnStar) + star;
                        i--;
                        posOnStar--;
                    }
                }
            }
            if (dec){
                star=prec.charAt(posOnStar)+star;
            }
        }
        System.out.println(star);
        return star;
    }

    public static boolean isParenthese(char a){
```

```
        if (a=='('||a==')') return true; else return false;
    }


     /**
     * Vérifie que le miRNA n'est pas complètement inclu dans la loop
     * @param prec
     * @param struct
     * @param mirna
     * @return true si le miRNA est dans la loop
     */
    public static boolean mirnaInLoop(String mirna, String prec, String struct){

        try {
            int start = prec.indexOf(mirna);
            int end = start + mirna.length();
            if (end > prec.length()||start==-1) {
                return false;
            }
            if (struct.substring(start, end).contains("(")&&struct.substring(start,
end).contains(")")) {
                return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

}
```

ii_checkMirnasStarsInCutadapt.java

Vérifie la présence des microARN star calculés précédemment dans le séquençage.

```
/*
 * Check presence of mirnas star in data after mapping
 */
package f_analyses;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.ObjectInputStream;
import java.io.PrintWriter;
import java.util.HashMap;
import tools.rnaobject;
import tools.tools;

/**
 *
 * @author Mickael
 */
public class ii_checkMirnasStarsInCutadapt {
    public static HashMap<String,String> hmstarsColor = new HashMap<String, String>();
    public static HashMap<String,Integer> hmstarsAbondance = new HashMap<String, Integer>();
    private static HashMap<String,Integer> hm = new HashMap<String, Integer>();


    //Executer d'abord
    //awk '{print $12}' FS="\t" all_lib.filters_analyses.com | perl -ne '$H{$_}++ or print' >
../forCheckMirnasStarsInCutadapt.txt
    public static void main(String args[]){
        String folderobjects="/ibrixfs1/Data/mik/cutadapt/objects/";
        File stars = new File(folderobjects+"mirnasStars.txt");
        File outfile = new File(folderobjects+"mirnasStarsInCutadapt.txt");
        createColorHashmap();
        addMirnasStarsToHashMap(stars);

//        for (int i = 1; i <= 10; i++) {
```

```
//            File infile = new
File("/ibrixfs1/Data/mik/cutadapt/cutadapt\\lib"+i+".allLengths.fasta");
//            checkpresence(infile);
//        }

        File infile = new
File("/ibrixfs1/Data/mik/cutadapt/mapping/all_lib.mapview.dump.mappedSequences.uni.txt");
        checkpresence(infile);

        //print hashmap
        System.out.println("Printing...")
                ;
        try {
            //
            PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile)));
            for (String s : hmstarsAbondance.keySet()) {
                pw.println(s+"\t"+hmstarsAbondance.get(s));
                System.out.println(s+"\t"+hmstarsAbondance.get(s));
            }

            pw.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    private static void checkpresence(File infile) {
        System.out.println("Checking mirnas stars in : "+infile);
        try {
            BufferedReader br = new BufferedReader(new FileReader(infile));
            String line = "";
            while(br.ready()){
                line=br.readLine();
                if (hmstarsColor.containsKey(line)){
                    String mirnaStar=hmstarsColor.get(line);
                    int tmp = hmstarsAbondance.get(mirnaStar);
                    tmp=tmp+1;
                    hmstarsAbondance.put(mirnaStar,tmp);
                }
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void addMirnasStarsToHashMap(File stars){
        System.out.println("Adding mirnasStars to Hashmap");
        try {
            BufferedReader br = new BufferedReader(new FileReader(stars));
            String line = "";
            while(br.ready()){
                line=br.readLine();
                String mirnaStar=line.split("\t")[3];
                if (mirnaStar.contains("-")){
                    mirnaStar=mirnaStar.substring(0, mirnaStar.length()-1);
                }
                mirnaStar=mirnaStar.replaceAll("U", "T");
                //TODO...
                String mirnaStarColor=getColorInATGC("T"+mirnaStar);

                hmstarsColor.put(mirnaStarColor,mirnaStar);
                hmstarsAbondance.put(mirnaStar, 0);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }



    private static String getColorInNumbers(String nt) {
        String color="";
        for (int i = 0; i < nt.length()-1; i++) {
            String pair = nt.substring(i, i+2);
```

```
            int c = hm.get(pair);
            color = color +c;
        }
        return "T"+color;
    }

    private static String getColorInATGC(String nt) {
        String color="";
        try {
            for (int i = 0; i < nt.length() - 1; i++) {
                String pair = nt.substring(i, i + 2);
                int c = hm.get(pair);

                if (c == 0) {
                    color = color + "A";
                }
                if (c == 1) {
                    color = color + "C";
                }
                if (c == 2) {
                    color = color + "G";
                }
                if (c == 3) {
                    color = color + "T";
                }
            }
        } catch (Exception e) {
        }

        return color;
    }

    private static void createColorHashmap() {
        hm.put("AA", 0);
        hm.put("AC", 1);
        hm.put("AG", 2);
        hm.put("AT", 3);
        hm.put("AN", 4);

        hm.put("CA", 1);
        hm.put("CC", 0);
        hm.put("CG", 3);
        hm.put("CT", 2);
        hm.put("CN", 4);

        hm.put("GA", 2);
        hm.put("GC", 3);
        hm.put("GG", 0);
        hm.put("GT", 1);
        hm.put("GN", 4);

        hm.put("TA", 3);
        hm.put("TC", 2);
        hm.put("TG", 1);
        hm.put("TT", 0);
        hm.put("TN", 4);

        hm.put("NA", 4);
        hm.put("NC", 4);
        hm.put("NG", 4);
        hm.put("NT", 4);
        hm.put("NN", 0);
    }

}
```

iii_conservedAgainstNCrnas.java

Ce script aurait pu être plus haut dans la hiérarchie puisqu'il sert à extraire les statistiques de blasts des petits ARNs mappés mais non prédits contre les non codants. Les non codants possèdent plusieurs catégories, et il faut déterminer combien d'ARN nous avons dans ces catégories.

```
/*
 * Extract statistics from blast against nonCode RNA
 *
```

```
 */
package f_analyses;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.HashMap;

/**
 *
 * @author Mickael
 */
public class iii_conservedAgainstNCrnas {

    public static void main(String args[]){
        File blastFile= new
File("/ibrixfs1/Data/mik/cutadapt/blasts/all_lib.smallrnas.blastncrna.txt");
        File trueblastFile = new
File("/ibrixfs1/Data/mik/cutadapt/blasts/all_lib.smallrnas.blastncrna.true.txt");
        File statsFile = new
File("/ibrixfs1/Data/mik/cutadapt/blasts/all_lib.smallrnas.blastncrna.stats.txt");
        File hmFile = new
File("/ibrixfs1/Data/mik/cutadapt/blasts/all_lib.smallrnas.blastncrna.hm.txt");

        HashMap<String,Integer> hm = new HashMap<String, Integer>();

        System.out.println("Reading blastFile");
        try {
            BufferedReader br = new BufferedReader(new FileReader(blastFile));
            PrintWriter pw = new PrintWriter(new FileWriter(trueblastFile));
            PrintWriter pwstatsFile = new PrintWriter(new FileWriter(statsFile));
            PrintWriter pwhmFile = new PrintWriter(new FileWriter(hmFile));

            String line = "";
            while (br.ready()){
                line = br.readLine();
                String infos[] = line.split(",");
                String qseqid = infos[0];
                String sallseqid = infos[1];
                int qstart = Integer.valueOf(infos[2])-1;
                int qend = Integer.valueOf(infos[3]);
                int sstart = Integer.valueOf(infos[4])-1;
                int send = Integer.valueOf(infos[5]);
                int length = Integer.valueOf(infos[8]);
                int gaps = Integer.valueOf(infos[12]);
                int mismatchs = Integer.valueOf(infos[11]);
                double pident = Double.valueOf(infos[9]);

                 //Calculate strand
                String strand;
                if (sstart<send){
                    strand = "+";
                } else strand = "-";

                //Calculate qcoverage (query coverage)
                double qcoverage = ((double)qend-(double)qstart)/(double)qseqid.length();

                //Determine conserved
                boolean conserved;

                if (pident>=80&&qcoverage>=1.0&&gaps<3){
                    conserved=true;
//                    if (sallseqid.contains("piRNA")&&gaps==0&&mismatchs==0){
//                        System.out.println(line.replaceAll(",",
"\t")+"\t"+strand+"\t"+qcoverage);
//                    }

                } else conserved=false;

                //print true blasts and add ncrnas hits to Hashmap
                if (conserved) {
                    pw.println(line.replaceAll(",", "\t")+"\t"+strand+"\t"+qcoverage);
                    String key="";
                    String tab[]= sallseqid.split("_");
```

```
        if (tab.length>4&&!sallseqid.contains("_NR")){
            key = qseqid + "\t" + tab[3] + "_" + tab[4];
        } else {
            key = qseqid+"\t"+sallseqid.substring(sallseqid.lastIndexOf("_")+1);
        }


        if (hm.containsKey(key)){
            int tmp = hm.get(key);
            tmp++;
            hm.put(key,tmp);
        } else hm.put(key, 1);
    }

}
br.close();
pw.close();

//RNA types
//more details here : http://www.noncode.org/index2.htm
int tmRNA=0;
int piRNA=0;
int snoRNA=0;
int mRNA=0;
int snmRNA=0;
int RNase=0;
int snRNA=0;
int SRP=0;
int gRNA=0;
int telomerase=0;
int selfsplicing=0;
int others=0;

System.out.println("Counting...");
for (String hit : hm.keySet()) {
    if (hit.contains("_")){
        hit=hit.substring(0,hit.indexOf("_"));
    }
    if (hit.contains("tmRNA")) tmRNA++;
    if (hit.contains("piRNA")) piRNA++;
    if (hit.contains("snoRNA")) snoRNA++;
    if (hit.contains("mRNAlike")) mRNA++;
    if (hit.contains("snmRNA")) snmRNA++;
    if (hit.contains("RNase")) RNase++;
    if (hit.contains("snRNA")) snRNA++;
    if (hit.contains("SRP")) SRP++;
    if (hit.contains("gRNA")) gRNA++;
    if (hit.contains("telomerase")) telomerase++;
    if (hit.contains("self-splicing")) selfsplicing++;
    others++;
    if(!hit.contains("tmRNA")&&!hit.contains("piRNA")&&
            !hit.contains("tmRNA")&&!hit.contains("piRNA")&&
            !hit.contains("snoRNA")&&!hit.contains("mRNAlike")&&
            !hit.contains("snmRNA")&&!hit.contains("RNase")&&
            !hit.contains("snRNA")&&!hit.contains("SRP")&&
            !hit.contains("gRNA")&&!(hit.contains("telomerase"))&&
            !hit.contains("self-splicing")){
        System.out.println(hit);
    }
    pwhmFile.println(hit+"\t"+hm.get(hit));
}

System.out.println("Statistics :");

System.out.println("tmRNA\t"+tmRNA);
pwstatsFile.println("tmRNA\t"+tmRNA);

System.out.println("piRNA\t"+piRNA);
pwstatsFile.println("piRNA\t"+piRNA);

System.out.println("snoRNA\t"+snoRNA);
pwstatsFile.println("snoRNA\t"+snoRNA);

System.out.println("mRNA\t"+mRNA);
pwstatsFile.println("mRNA\t"+mRNA);
```

```
            System.out.println("snmRNA\t"+snmRNA);
            pwstatsFile.println("snmRNA\t"+snmRNA);

            System.out.println("RNase\t"+RNase);
            pwstatsFile.println("RNase\t"+RNase);

            System.out.println("snRNA\t"+snRNA);
            pwstatsFile.println("snRNA\t"+snRNA);

            System.out.println("SRP\t"+SRP);
            pwstatsFile.println("SRP\t"+SRP);

            System.out.println("selfsplicing\t"+selfsplicing);
            pwstatsFile.println("selfsplicing\t"+selfsplicing);

            System.out.println("telomerase\t"+telomerase);
            pwstatsFile.println("telomerase\t"+telomerase);

            int all =
tmRNA+piRNA+snoRNA+mRNA+snmRNA+RNase+snRNA+SRP+gRNA+telomerase+selfsplicing;
            others=others-all;

            System.out.println("Others\t"+others);
            pwstatsFile.println("Others\t"+others);

            pwstatsFile.close();
            pwhmFile.close();

        } catch (Exception e) {
            e.printStackTrace();
        }

    }

}
```

g_families

checkNumberFamiliesMiRbase.java

Ce script compte le nombre de familles présentes dans miRbase. Ce repère du nombre de familles est important pour le script suivant.

```
/*
 * Get amount of families in mirbase
 */
package g_families;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.HashMap;

/**
 *
 * @author Mickael
 */
public class checkNumberFamiliesMiRbase {
    public static HashMap<String,Integer> hm = new HashMap<String, Integer>(20000);

    public static void main(String args[]){
        File mirnas = new
File("H:\\mik\\_version3\\vii_blast\\databases\\miRbase.mirnasT.fasta");

        try {
            BufferedReader br = new BufferedReader(new FileReader(mirnas));
            String line="";
            while (br.ready()){
                line = br.readLine().toLowerCase();
                if (line.startsWith(">")){
                    String family = null;
                    try {
                        family = line.substring(line.lastIndexOf("mir")).replace("mir",
"").replace("-", "").
```

```
                                replace("*", "").replace("5p", "").replace("3p",
"").replaceAll("[a-z]", "").
                                replace(".1","").replace(".2","").replace(".3","");
                        //System.out.println(family);
                        if (hm.containsKey(family)){
                            int tmp=hm.get(family);
                            tmp++;
                            hm.put(family, tmp);
                        } else {
                            hm.put(family, 1);
                        }
                    } catch (Exception e) {
                        //System.err.println(line);
                    }
                }
            )

            br.close();


            System.out.println("Family\tOccurences");
            for (String s : hm.keySet()) {
                System.out.println(s+"\t"+hm.get(s));
            }
            System.out.println("Amount of families:"+hm.size());
        } catch (Exception e) {
            e.printStackTrace();
        }


    }

}
```

getFamiliesFromClustalTree.java

Les microARNs prédits doivent être classés en familles. Pour celà, on opère un alignement multiple sur toutes les sequences avec Clustalw.L'arbre fourni est ensuite lu par ce script. Chaque branche de l'arbre contient une mesure de distance, et c'est cette dernière qui permet de savoir si on reste dans le même clade ou non. La mesure de distance limite entre "rester dans le même clade" ou "changer de clade" est mesurée sur un test positif sur miRbase dans le script précédent.

```
/*
 * Get groups with treshold from a newick tree
 */
package g_families;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;

/**
 *
 * @author Mickael
 */
public class getFamiliesFromClustalTree {

    public static double limit = 0.153;
    public static String tree="";
    public static int i;
    public static int j;

    public static void main(String args[]){
        String familyobjects="/ibrixfs1/Data/mik/cutadapt/families/";

        //nos mirnas
        File intree = new
File(familyobjects+"all_lib.pred.analyses.com.uni.mirnas+mirbase.fasta.tree");
        File outfile = new
File(familyobjects+"all_lib.pred.analyses.com.uni.mirnas+mirbase.fasta.tree.group");
```

```
        System.out.println("limit"+"\tTotal"+"\tgroups");

        for (limit = 0.15285; limit <= 0.15285; limit=limit+0.0001) {
            cluster(intree, outfile);
            tree="";
        }
    }

    private static void cluster(File intree, File outfile) {
        //System.out.println("Clustering...");
        PrintWriter pw=null;
        BufferedReader br=null;
        //(A:0.0,B:0.0):0.1):0.2,C:0.3)
        int cpt=0;
        int group=0;
        try {
            br = new BufferedReader(new FileReader(intree));
             pw = new PrintWriter(new FileWriter(outfile));

            while(br.ready()){
                tree += br.readLine().trim();
            }


            group=1;

            for (i = 0; i < tree.length(); i++) {
                if (tree.charAt(i)=='('){
                    //i++;
                } else {
                    //bloc '(A:0.0,'
                    String clade=getStringFromTree(',');
                    String name = clade.split(":")[0];
                    double dbranch = Double.valueOf(clade.split(":")[1]);
                    //System.out.println(name+"\t"+group);
                    pw.println(name.replace(",", "") +"\t"+group);
                    cpt++;
                    if (dbranch>limit) {
                        group++;
                    }

                    i++;
                    while (tree.charAt(i)!='(') {
                        //bloc ',B:0.0):0.1):0.2,C:0.3)'

                        //recuperation de B
                        name=getStringFromTree(':');
                        i++;
                        //recuperation de 0.0
                        String sbranch=getStringFromTree(')');
                        i++;
                        dbranch=Double.valueOf(sbranch);
                        //affichage B
                        //System.out.println(name+"\t"+group);
                        pw.println(name.replace(",", "")+"\t"+group);
                        cpt++;
                        boolean groupplus=false;
                        if (dbranch>limit) {
                            group++;
                            groupplus=true;
                        }
                        //passage de ':'
                        i++;
                        //recuperation de 0.1 et suivant
                        sbranch=getStringFromTree(',');

                        //bloc ',B:0.0):0.1):0.2,C:0.3)'
                        //Cas de 0.2
                        if (sbranch.contains("):")){
                            String beforeOpenPar=getStringFromTree('(');

                            // cas C et suivants
                            if (containsBase(beforeOpenPar)){
                                j=1;
                                while (j!=beforeOpenPar.length()){
```

```
                                if (containsBase(beforeOpenPar.substring(j))) {
                                    name = getStringFromTree2(beforeOpenPar, ':');
                                    j++;
                                    //recuperation de 0.0
                                    sbranch = getStringFromTree2(beforeOpenPar, ')');
                                    j++;
                                    dbranch = Double.valueOf(sbranch);
                                    //affichage B
                                    //System.out.println(name + "\t" + group);
                                    pw.println(name.replace(",", "") + "\t" + group);
                                    cpt++;
                                    groupplus = false;
                                    if (dbranch > limit) {
                                        group++;
                                        groupplus = true;
                                    }
                                    //passage de ':'
                                    j++;
                                    //recuperation de 0.1 et suivant
                                    sbranch = getStringFromTree2(beforeOpenPar, ',');

                                } else j++;
                            }
                        }
                        if(!groupplus){
                            group++;
                        }
                        //cas normal
                    } else {
                        dbranch=Double.valueOf(sbranch);
                        if (dbranch>limit&&!groupplus) {
                            group++;
                        }
                        i++;
                    }

                }
            }
        }
        br.close();
        pw.close();
    } catch (Exception e) {
        //e.printStackTrace();
        pw.close();
    }
    //System.out.println("limit:"+limit+"\tTotal: "+cpt+"\tgroups: "+group);
    System.out.println(limit+"\t"+cpt+"\t"+group);
}

public static boolean containsBase(String s){
    if (s.contains("A")||s.contains("U")||s.contains("G")||s.contains("C")){
        return true;
    }else return false;

}

private static String getStringFromTree(char c) {
    String s="";
    while(tree.charAt(i)!=c){
        s+=tree.charAt(i);
        i++;
    }
    return s;
}

private static String getStringFromTree2(String treepart, char c) {
    String s="";
    while(treepart.charAt(j)!=c){
        s+=treepart.charAt(j);
        j++;
    }
    return s;
}
}
```

tools

i_addMaqResultsToObject.java

Ce script regroupe les résultats de sortie de MAQ pour les stocker dans des objets. Script execute après tous ceux de a_cutadapt

```
/*
 * Add all results from maq to object per size of miRNAs, so 15 * 10 objects
 *  (15 to 30 for size, so 15, and 10 librairies)
 */

package tools;

import java.io.File;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Vector;

/**
 *
 * @author Mickael
 */
public class i_addMaqResultsToObject {
    private static int cpt=1;
    public static boolean debug = true;
    public static HashMap<String,String> hmfastq;
    public static HashMap<String,String> hmexprVirg;    //avec virgules, toutes les expressions
issu de iii_getExpression
    public static HashMap<String,String[]> hmexprTab;   //avec tabulation, fichier plus petit,
issu de iv_adaptexpression.java
    public static HashMap<String,Vector> hmdump= new HashMap<String,Vector>();

    public static void main(String args[]){
        tools.createTranslateColorHashmap();
        addResultstoObjectPerSize();
    }

    /**
     * Add all results to object per size of miRNAs, so 15 * 10 objects
     *  (15 to 30 for size, so 15, and 10 librairies)
     */
    public static void addResultstoObjectPerSize(){
       String f = "/ibrixfs1/Data/mik/cutadapt/mapping/";

        try {
            //stockage du fichier d'expressions
            File expressionFile = new
File("/ibrixfs1/Data/mik/cutadapt/fasta/all_lib.smallrna.uni.expression.tab.tot.sup5.readNames.tx
t");
            System.out.println("Adding expression file to hashmap...");
            addExprToHashmapTab(expressionFile);
            //traitement
            for (int i = 1; i <= 10; i++) {
                for (int j = 15; j <= 30; j++) {

                    File rnaObjectFile = new File("/ibrixfs1/Data/mik/cutadapt/objects/" + i
+ "/lib" + i + "_" + j + ".obj");
                    File outfile = new File("/ibrixfs1/Data/mik/cutadapt/objects/" + i +
"/lib" + i + "_" + j + ".txt");
                    if (rnaObjectFile.exists()) {
                        rnaObjectFile.delete();
                    }

                    if(outfile.exists()) outfile.delete();
                    ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(rnaObjectFile, true));
```

```
                        PrintWriter pw = new PrintWriter(new BufferedWriter(new
FileWriter(outfile, true)));
                        rnaobject2 rtmp= new rnaobject2();
                        pw.println(rtmp.toStringHeader());

                        File fastqFile = new File(f + i + "/lib" + i + "_" + j + ".fastq");
                        File mapviewFile = new File(f + i + "/lib" + i + "_" + j +
".genomeble6db.mapview.seq");   //output from vii_getSequencesPrecursorsFromMapview
                        File dumpout = new File(f + i + "/lib" + i + "_" + j +
".genomeble6db.dump.out.seq");  //output from vi_getSequencesPrecursorsFromDump
                        System.out.println("lib" + i + "_" + j);
                        processResults(j, fastqFile, mapviewFile, dumpout, rnaObjectFile,
oos,pw);

                        //add last object at id=0
                        rnaobject2 r = new rnaobject2();
                        r.setId(0);
                        oos.writeObject(r);


                        oos.flush();
                        //oos.reset();
                        oos.close();

                    )

                )

            }catch(Exception e){
                e.printStackTrace();
            )
        }

        /**
         * Main fonction
         * @param fastqFile
         * @param mapviewFile
         * @param dump
         * @param rnaObjectFile
         * @param oos
         */
        public static void processResults(int taille,File fastqFile, File mapviewFile, File dump,
                File rnaObjectFile,ObjectOutputStream oos,PrintWriter pw) {

            //System.out.println("Reading mapview File");

            int compteur=0;
            try {
                addFastqToHashmap(fastqFile);
                //addDumpToHashmap(dump);
                //reading mapviewFile
                BufferedReader br = new BufferedReader(new FileReader(mapviewFile));
                String line = "";
                while (br.ready()){
                    line = br.readLine();
                    String tab[] = line.split("\t");
                    if (hmexprTab.containsKey(tab[0])){
                        rnaobject2 r = new rnaobject2();
                        r.setId(cpt);

                        // parsing mapviewFile fields
                        r.setSmallRNAcolorName(tab[0]);
                        r.setMappingSourceAccession(tab[1]);
                        r.setStartPositionOnSource(Integer.valueOf(tab[2]));
                        r.setStrandOnSource(tab[3]);
                        r.setMappingMismatchs(Integer.valueOf(tab[9]));
                        r.setSmallRNAsequenceAfterMapping(tab[16]);
                        //r.setMappingSourceName(tab[17]);
                        r.setPrecursorSequence(tab[18]);
                        r.setMapper("mapview");

                        //parsing fastqFile to get the original sequence
                        r.setOriginalSmallRNAsequence(hmfastq.get(r.getSmallRNAcolorName()));
                        //Expression normale sans les reads
                        int expr[]=new int[10];
                        int tot = 0;
```

```
                //Expression normale avec les reads

                try {
                      String exp[]=hmexprTab.get(r.getSmallRNAcolorName());
                      if (exp!=null){

                          for (int j = 0; j < exp.length-1; j++) {
                              expr[j] = Integer.valueOf(exp[j]);
                          }
                          tot=Integer.valueOf(exp[10]);
                          r.setExpressionPerLib(expr);
                          r.setExpressionTotal(Integer.valueOf(tot));
                      }
                      else {
                          r.setExpressionTotal(0);
                      }

                } catch (Exception ex) {
                    if (debug) ex.printStackTrace();
                }


                //Ecriture
                writeObjectUnderConditions(taille,r,oos,pw);


                if (compteur%100000==0){
                    System.out.print("*");
                    oos.flush();
                    pw.flush();
                }
                cpt++;
                compteur++;

                /**
                 * dump (multiple hits)
                 * tab d'un dump :
nom,smallrna,strand,mismatch,accession,sourcename,précurseur
                 *
                 * On continue sur le smallRNA que l'on traitait juste avant
                 *
                 */
                //Finalement le dump ne sera pas prit en compte
//                  try {
//                      Vector v = new Vector();
//                      v = hmdump.get(r.getSmallRNAcolorName());
//                      if (v!=null) {
//                          int max=v.size();
//                          if(max>=20){
//                              max=20;
//                          }
//                          for (int i = 0; i < max; i++) {
//                              //reprise des informations de l'objet qui vient d'être créé
//                              r = new rnaobject2();
//                              r.setId(cpt);
//                              r.setSmallRNAcolorName(tab[0]);
//                              r.setMappingSourceAccession(tab[1]);
//                              r.setStartPositionOnSource(Integer.valueOf(tab[2]));
//                              r.setStrandOnSource(tab[3]);
//                              r.setMappingMismatchs(Integer.valueOf(tab[9]));
//                              r.setSmallRNAsequenceAfterMapping(tab[14]);
//                              String origName=hmfastq.get(r.getSmallRNAcolorName());
//                              r.setOriginalSmallRNAsequence(origName);
//                              r.setExpressionPerLib(expr);
//                              r.setExpressionTotal(tot);
//                              r.setMapper("dump");
//                              //insertion du dump
//                              String s[] = (String[]) v.get(i);
//
//                              r.setSmallRNAsequenceAfterMapping(s[1]);
//                              r.setStrandOnSource(s[2]);
//                              r.setMappingMismatchs(Integer.valueOf(s[3]));
//                              r.setMappingSourceAccession(s[4]);
//                              //r.setMappingSourceName(s[5]);
```

```
//                                r.setPrecursorSequence(s[6]);
//                                writeObjectUnderConditions(taille, r, oos,pw);
//                                cpt++;
//                                compteur++;
//                            }
//                        }
//
//                    } catch (Exception e) {
//                        if (debug) e.printStackTrace();
//                    }
                }
            }

            br.close();
            System.out.println("Objects added : "+compteur);
        } catch (Exception e) {
            if (debug) e.printStackTrace();
        }

    }


    public static String addFastqToHashmap(File fastqFile){
        //System.out.println("Add fastq to hashmap");
        String s="";
        //System.out.print("Check number of sequence in "+fastqFile.getName()+"...");
        int reads_size = countSequencesInFastq(fastqFile);
        //System.out.println(reads_size);
        int i=0;
        //System.out.print("Get Reads in hashmap...");
        hmfastq = new HashMap<String,String>(reads_size+100,10000);
        try {
            BufferedReader br = new BufferedReader(new FileReader(fastqFile));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                if (line.startsWith("@")) {
                    String seq = br.readLine();
                    hmfastq.put(line.substring(1), seq);
                    br.readLine();
                    br.readLine();

//                    String seq = br.readLine();
//                    if
(seq.startsWith("A")||seq.startsWith("T")||seq.startsWith("G")||seq.startsWith("C")){
//                        hmfastq.put(line.substring(1), seq);
//                    }
                }

            }
        } catch (Exception ex) {
            System.err.println("Error in addFastqToHashmap at "+i+" on "+reads_size);
        }
        //System.out.println("Finished");

        return s;
    }

    /**
     * Calcule le nombre de reads
     * @param reads
     * @return nombre de reads
     */
    public static int countSequencesInFastq(File fastq) {
        int count=0;
        try {
            BufferedReader br=new BufferedReader(new FileReader(fastq));
            while (br.ready()) {
                if (br.readLine().startsWith("@")){
                    br.readLine();
                    br.readLine();
                    br.readLine();
                    count++;
                }
            }
```

```
            br.close();
         ) catch (Exception e) {
            System.out.println(e.getMessage());
            return 1000000;
        }
        return count;
    }

     /**
     * Adapté pour le format tabulation
     * sequence\t1\t2\t3\t4\t5\t6\t7\t8\t9\t10\ttotal\nreads
     * @param expressionFile
     */
    private static void addExprToHashmapTab(File expressionFile) {
        int reads_size = tools.countLines(expressionFile);

        int i=0;

        hmexprTab = new HashMap<String,String[]>(70000000);
        try {
            BufferedReader br = new BufferedReader(new FileReader(expressionFile));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                String tab[]=line.split("\t");
                String seq = tab[0];
                String
exp[]={tab[1],tab[2],tab[3],tab[4],tab[5],tab[6],tab[7],tab[8],tab[9],tab[10],tab[11]};

                String reads[]=tab[12].split(",");

                for (String read : reads) {
                    hmexprTab.put(read+"/1",exp);
                }

            }
        } catch (Exception ex) {
            System.err.println("Error in addExprHashmap at "+i+" on "+reads_size);
        }
    }

    /**
     * Adapté pour le format
     * sequence\t1,2,3,4,5,6,7,8,9,10
     * @param expressionFile
     */
    private static void addExprToHashmapvirg(File expressionFile) {
        int reads_size = tools.countLines(expressionFile);

        int i=0;

        hmexprVirg = new HashMap<String,String>(reads_size+100,10000);
        try {
            BufferedReader br = new BufferedReader(new FileReader(expressionFile));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                hmexprVirg.put(line.split("\t")[0], line.split("\t")[1]);
                if (i%10000000==0){
                    System.out.print("*");
                }
            }
        } catch (Exception ex) {
            System.err.println("Error in addExprHashmap at "+i+" on "+reads_size);
        }
    }

    private static void addDumpToHashmap(File dump) {
        //System.out.println("Add dump to hashmap");
        int reads_size = tools.countLines(dump);
        int i=0;
        //System.out.print("Get Reads in hashmap...");
        hmdump = new HashMap<String,Vector>(reads_size+100,10000);
        try {
```

```
            BufferedReader br = new BufferedReader(new FileReader(dump));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                String s[]=line.split("\t");
                Vector v = new Vector();
                v.add(s);
                Object obj = hmdump.get(s[0]);
                if (obj!=null) {
                    v=hmdump.get(s[0]);
                    v.add(s);
                    hmdump.put(s[0], v);
                } else {
                    hmdump.put(s[0], v);
                }

                if (i%10000000==0){
                    System.out.print("*");
                }
            }
        } catch (Exception ex) {
            System.err.println("Error in addDumpToHashmap at "+i+" on "+reads_size);
        }
    }

    /**
     * Ecrit l'objet avec des conditions
     * @param r
     * @param oos
     * @return
     */
    private static boolean writeObjectUnderConditions(int taille, rnaobject2 r,
ObjectOutputStream oos,PrintWriter pw) {
        boolean b=true;
        try {
            r.getSmallRNAsequenceAfterMappingWithNT();
            if (r.getExpressionTotal()>=5) {
                if (taille <= 17) {
                    if (r.getMappingMismatchs() <= 1) {
                        oos.writeObject(r);
                        pw.println(r.toString());
                    }
                } else {
                    if (r.getMappingMismatchs() <= 2) {
                        oos.writeObject(r);
                        pw.println(r.toString());
                    }
                }
            }
        } catch (IOException iOException) {
            iOException.printStackTrace();
            b=false;
        }
        return b;
    }

}
```

ii_insertPredictionInObjects.java

Script à exécuter après les prédictions. Il met à jour les objets précédemment créés pour y ajouter les prédictions.

```
/*
 * Récupère les objets après mapping
 * Ecrit de nouveaux objets avec les résultats de HHMMiR et miPred
 * Si un id est détécté dans les prédictions, un objet est créé, sinon non
 */

package tools;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;


/**
 * Avant de lancer ce script :
 * cd /cygdrive/h/mik/_version3
 * for i in {1..10};do cp i_mapping/cutadapt/$i/*.obj objets/$i/;done
 * @author Mickael
 */
public class ii_insertPredictionInObjects {

    public static String objectsFolder="/ibrixfs1/Data/mik/cutadapt/objects/";
    public static String
HHMMiRpredictionFolder="/ibrixfs1/Data/mik/cutadapt/prediction/hhmmir_results/";
    public static String
miPredpredictionFolder="/ibrixfs1/Data/mik/cutadapt/prediction/mipred_results/";
    public static HashMap<String,ArrayList> hmMipred ;
    public static HashMap<String,ArrayList> hmHHMMiR ;

    public static void main(String args[]){
        addPredictionsToObjects();
        iii_addTargetsAndAnalysesToObjects.main(args);
    }

    /**
     * Ajoute les predictions de mipred aux objets
     */
    public static void addPredictionsToObjects(){
        tools.createTranslateColorHashmap();
        for (int i = 1; i <=10; i++) {
            for (int j = 15; j <= 30; j++) {
                File INobjFile= new File(objectsFolder+i+"/lib"+i+"_"+j+".obj");
                File OUTobjFile= new File(objectsFolder+i+"/lib"+i+"_"+j+".pred.obj");

                File miPredPredictionFile = new
File(miPredpredictionFolder+i+"/lib"+i+"_"+j+".smallRNAs.mapped.folded.forPrediction.mipred.forRF
.RF");
                File HHMMiRPredictionFile = new
File(HHMMiRpredictionFolder+i+"/lib"+i+"_"+j+".smallRNAs.mapped.folded.forPrediction.hairpins.hhm
mir0995.true");
                if (OUTobjFile.exists()) OUTobjFile.delete();

                // Fichier contenant toutes les informations sur les miRNAs prédits
                File outfile= new File(objectsFolder+i+"/lib"+i+"_"+j+".pred");
                if (outfile.exists()) outfile.delete();
                // Fichier des communs entre hhmmir et mipred
                File outfileCom= new File(objectsFolder+i+"/lib"+i+"_"+j+".com");
                if (outfileCom.exists()) outfileCom.delete();

                System.out.println("Reading file "+INobjFile.getName());
                int cpt=0;
                rnaobject2 r=null;
                ObjectInputStream ois=null;
                ObjectOutputStream oos=null;
                PrintWriter pw = null;
                //PrintWriter pwExp = null;
                PrintWriter pwCom = null;
                try {
                    addMipredPredictionsToHashMap(miPredPredictionFile);
                    addHHMMiRPredictionsToHashMap(HHMMiRPredictionFile);

                    ois = new ObjectInputStream(new FileInputStream(INobjFile));
                    oos = new ObjectOutputStream(new FileOutputStream(OUTobjFile, true));
                    pw = new PrintWriter(new BufferedWriter(new FileWriter(outfile, true)));
                    //pwExp = new PrintWriter(new BufferedWriter(new FileWriter(outfileExp,
true)));
```

```
                    pwCom = new PrintWriter(new BufferedWriter(new FileWriter(outfileCom,
true)));

                    boolean b=true;
                    while (b) {
                        r = (rnaobject2) ois.readObject();
                        //teste fin du fichier pour éviter une exception
                        if (r.getId()==0){
                            b = false;
                        } else {

                            if (r.getId()==4089088){
                                System.out.print("");
                            }


                            boolean mirnaInPrec=true;
                            //imprime miPred
                            String mirna = r.getSmallRNAsequenceAfterMapping();
                            ArrayList<String> mipredlist = new ArrayList<String>();
                            mipredlist = hmMipred.get(mirna);
                            String mipred=null;
                            if (mipredlist!=null){
                                ArrayList<String> mipredlisttmp = new ArrayList<String>();
                                for (String string : mipredlist) {
                                    mipredlisttmp.add(string);
                                }
                                //get the best precursor sequence score
                                mipred = getBestPrecInmiPredPrecurors(mirna,mipredlisttmp);
                                //si tmp n'est pas vide
                                if (mipred!=null){
                                    String tab[] = mipred.split("\t");
                                    r.setPrecursorSequence(tab[1]);
                                    r.setPrecursorStructure(tab[4]);
                                    r.setPrecursorStructureMFE(Double.valueOf(tab[5]));
                                    r.setPredictorName("mipred");
                                    r.setPredictorScore(tab[7]);    //p-value (shuffle
times:1000)
                                    r.setMipredRealpseudo(tab[8]);
                                    r.setMipredConfidence(Double.valueOf(tab[9].replace("%",
"")));
                                    r.setMiRNA(true);

                                    if (tab[1].contains(mirna)){
                                        oos.writeObject(r);
                                        pw.println(r.toString());
                                    } else {
                                        mirnaInPrec=false;
                                    }
                                } else {
                                    mirnaInPrec=false;
                                }
                            }

                            // imprime HHMMiR

                            ArrayList<String> hhmmirlist = hmHHMMiR.get(mirna);
                            boolean eq = false;
                            if (hhmmirlist!=null){
                                ArrayList<String> hhmmirlisttmp = new ArrayList<String>();
                                for (String string : hhmmirlist) {
                                    hhmmirlisttmp.add(string);
                                }
                                //get the best precursor sequence score
                                String hhmmir =
getBestPrecInHHMMiRPrecurors(mirna,hhmmirlisttmp);
                                //si tmp n'est pas vide
                                if (hhmmir!=null){
                                    String tab[] = hhmmir.split(",");
                                    //verifie que le mirna est dans le precurseur
                                    if (!tab[1].toUpperCase().contains(mirna)){
                                        mirnaInPrec=false;
                                    }

                                    //si commun on imprime l'objet avec mipred dans le fichier
des communs
```

```
                                    if (mipred!=null&&mirnaInPrec){
                                        pwCom.println(r.toStringCommons());
                                        eq=true;
                                    }
                                    //on refait un objet si il y a un commun
                                    if (eq){
                                        int id = r.getId();
                                        String SmallRNAcolorName = r.getSmallRNAcolorName();
                                        int ExpressionPerLib[] = r.getExpressionPerLib();
                                        int getExpressionTotal = r.getExpressionTotal();
                                        String OriginalSmallRNAsequence =
r.getOriginalSmallRNAsequence();
                                        String Mapper = r.getMapper();
                                        String MappingSourceAccession =
r.getMappingSourceAccession();
                                        int MappingMismatchs = r.getMappingMismatchs();
                                        String MappingSourceName = r.getMappingSourceName();
                                        int StartPositionOnSource = r.getStartPositionOnSource();
                                        String StrandOnSource = r.getStrandOnSource();
                                        String SmallRNAsequenceAfterMapping =
r.getSmallRNAsequenceAfterMapping();

                                        r = new rnaobject2();
                                        r.setId(id);
                                        r.setSmallRNAcolorName(SmallRNAcolorName);
                                        r.setExpressionPerLib(ExpressionPerLib);
                                        r.setExpressionTotal(getExpressionTotal);
                                        r.setOriginalSmallRNAsequence(OriginalSmallRNAsequence);
                                        r.setMapper(Mapper);
                                        r.setMappingSourceAccession(MappingSourceAccession);
                                        r.setMappingMismatchs(MappingMismatchs);
                                        r.setMappingSourceName(MappingSourceName);
                                        r.setStartPositionOnSource(StartPositionOnSource);
                                        r.setStrandOnSource(StrandOnSource);

r.setSmallRNAsequenceAfterMapping(SmallRNAsequenceAfterMapping);

                                    }

                                    r.setPrecursorSequence(tab[1]);
                                    r.setPrecursorStructure(tab[2]);
                                    r.setPrecursorStructureMFE(0);
                                    r.setPredictorName("HHMMiR");
                                    r.setPredictorScore(tab[5]);    //Model score
                                    r.setNotes(tab[3]+"\t"+tab[4]); //Prediction result (real or
pseudo) and Prediction confidence(%)
                                    r.setMiRNA(true);


                                    if (mirnaInPrec) {
                                        oos.writeObject(r);
                                        pw.println(r.toString());
                                    }
                                    //si commun on imprime l'objet avec hhmmir dans le fichier
des communs
                                    if (eq) {
                                        pwCom.println(r.toStringCommons());
                                    }
                                }


                            }
                            eq=false;
                        }
                        if (cpt%10000==0){
                            System.out.print("*");
                            oos.flush();
                            pw.flush();
                            //pwExp.flush();
                            pwCom.flush();
                        }
                        cpt++;
                    }
                    //add last object at id=0 to avoid exception at reading
                    r = new rnaobject2();
                    r.setId(0);
```

```
                oos.writeObject(r);


                oos.close();
                pw.close();
                ois.close();
                //pwExp.close();
                pwCom.close();

                System.out.println("Finished");
            } catch(Exception e) {
                try {
                    oos.close();
                } catch (IOException ex) {
                }
                e.printStackTrace();
            }
        }
    }
}

    private static String getBestPrecInmiPredPrecurors(String mirna,ArrayList<String> mipred) {
        //Si un seul precurseur dans la liste on le retourne
        if (mipred.size()==1){
            return mipred.get(0);
        }


        //verifie que le mirna est dans le precurseur pour toute la liste de mipred
        int removed=0;
        for (int i = 0; i< mipred.size();i++) {
            String tab[] = mipred.get(i).split("\t");
            String prec= tab[1];
            if (!prec.contains(mirna)){
                mipred.remove(i);
                mipred.add(i, "");
                removed++;
            }
        }
        //verifie que tout n'a pas ete enleve dans la liste de mipred
        if (mipred.size()!=removed){
            //Mets les scores dans une liste de score
            ArrayList<Double> scores= new ArrayList<Double>();
            for (String s : mipred) {
                if (!s.isEmpty()) {
                    String tab[] = s.split("\t");
                    String score = tab[7];
                    scores.add(Double.valueOf(score));
                }

            }
            //tri de la liste de score
            Collections.sort(scores);

            //recupere dans la liste de mipred celui qui a le meilleur score, celui classe en
premier dans la liste de scores
            for (String s : mipred) {
                if (s.contains(scores.get(0).toString())){
                    return s;
                }
            }
        } else {
            return null;
        }

        return null;
    }

    private static String getBestPrecInHHMMiRPrecurors(String mirna,ArrayList<String> hhmmir) {

        //Si un seul precurseur dans la liste on le retourne
        if (hhmmir.size()==1){
            return hhmmir.get(0);
        }
```

```
        //verifie que le mirna est dans le precurseur pour toute la liste de hhmmir
        int removed=0;
        for (int i = 0; i< hhmmir.size();i++) {
            String tab[] = hhmmir.get(i).split(",");
            String prec= tab[1];
            if (!prec.contains(mirna)){
                hhmmir.remove(i);
                hhmmir.add(i, "");
                removed++;
            }
        }
        //verifie que tout n'a pas ete enleve dans la liste de mipred
        if (hhmmir.size()!=removed){
            //Mets les scores dans une liste de score
            ArrayList<Double> scores= new ArrayList<Double>();
            for (String s : hhmmir) {
                if (!s.isEmpty()) {
                    String tab[] = s.split(",");
                    String score = tab[5];
                    scores.add(Double.valueOf(score));
                }
            }
            //tri de la liste de score
            Collections.sort(scores);

            //recupere dans la liste de mipred celui qui a le meilleur score, celui classe en
premier dans la liste de scores
            for (String s : hhmmir) {
                if (s.contains(scores.get(0).toString())){
                    return s;
                }
            }
        } else {
            return null;
        }

        return null;
    }

/**
 * Ajoute le fichier de prediction
 * @param miPredPredictionFile
 */
    private static void addMipredPredictionsToHashMap(File miPredPredictionFile) {
        int countlines = tools.countLines(miPredPredictionFile);
        int i=0;
        hmMipred = new HashMap<String,ArrayList>(countlines+100,100);

        try {
            BufferedReader br = new BufferedReader(new FileReader(miPredPredictionFile));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                String mirna = line.split("\t")[0].toUpperCase().replace("T","U");
                ArrayList<String> al = new ArrayList<String>();
                al.add(line);
                if (hmMipred.containsKey(mirna)){
                    ArrayList<String> tmp=hmMipred.get(mirna);
                    tmp.add(line);
                    hmMipred.put(mirna, tmp);
                } else {
                    hmMipred.put(mirna, al);
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in addExprHashmap at "+i+" on "+countlines);
        }
    }

    private static void addHHMMiRPredictionsToHashMap(File HHMMiRPredictionFile) {
        int countlines = tools.countLines(HHMMiRPredictionFile);
        int i=0;
```

```
        hmHHMMiR = new HashMap<String,ArrayList>(countlines+100,100);
        try {
            BufferedReader br = new BufferedReader(new FileReader(HHMMiRPredictionFile));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                int u=line.indexOf("_");
                String mirna = line.split("\t")[0].substring(1,u).toUpperCase().replace("T","U");
                ArrayList<String> al = new ArrayList<String>();
                al.add(line);
                if (hmHHMMiR.containsKey(mirna)){
                    ArrayList<String> tmp=hmHHMMiR.get(mirna);
                    tmp.add(line);
                    hmHHMMiR.put(mirna, tmp);
                } else {
                    hmHHMMiR.put(mirna, al);
                }
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in addExprHashmap at "+i+" on "+countlines);
        }
    }

}
```

iii_addTargetsAndAnalysesToObjects.java

Ce script est à exécuter après les gènes cibles. Il met à jour tous les objets après prédiction. Il regroupe de nombreuses autres analyses telles que : ribosomaux, conservés, groupes, non codants, faible complexité, microARN*, familles, qualité des reads, MFE des précurseurs recalculés, les expressions concaténées, et le gene ontologies.

```
/*
 *  Met à jour les objets, et vérifie qu'il n'y a pas de ribosomaux et qu'il y a des target genes
 */

package tools;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;
import java.util.HashMap;

/**
 *
 * @author Mickael
 */
public class iii_addTargetsAndAnalysesToObjects {

    public static HashMap<String,String> hmrefs = new HashMap<String,String>();
    public static HashMap<String,String> hmrib = new HashMap<String,String>();
    public static HashMap<String,String> hmtargets = new HashMap<String,String>();
    public static HashMap<String,String> hmgroups = new HashMap<String,String>();
    public static HashMap<String,String> hmconserved = new HashMap<String,String>();
    public static HashMap<String,String> hmNonCodants = new HashMap<String,String>();
    public static HashMap<String,String> hmLC = new HashMap<String,String>();
    public static HashMap<String,String> hmstars = new HashMap<String,String>();
    public static HashMap<String,Integer> hmstarsCutadapt = new HashMap<String,Integer>();
    public static HashMap<String,Integer> hmfamily = new HashMap<String,Integer>();
    public static HashMap<String,String> hmreadqual = new HashMap<String,String>();
    public static HashMap<String,Integer> hmmirnaPrec = new HashMap<String,Integer>();
    public static HashMap<String,Double> hmMFEPrec = new HashMap<String,Double>();
    public static HashMap<String,int[]> hmexp = new HashMap<String,int[]>();
    public static HashMap<String,String> hmgoP = new HashMap<String,String>();
```

```
    public static HashMap<String,String> hmgoC = new HashMap<String,String>();
    public static HashMap<String,String> hmgoF = new HashMap<String,String>();
    public static HashMap<String,String> hmgoProtName = new HashMap<String,String>();
    public static HashMap<String,String> hmgoProtID = new HashMap<String,String>();

    public static String folder="/ibrixfs1/Data/mik/cutadapt/";
    public static String folderobjects="/ibrixfs1/Data/mik/cutadapt/objects/";

    public static void main(String args[]){
        tools.createTranslateColorHashmap();
        File refsFile=new File("/ibrixfs1/Data/mik/genome_ble6DB_names.txt");
        addRefsToHashMap(refsFile);

        File ribFile = new File
(folder+"blasts/ribosomaux/all_lib.pred.analyses.com.uni.mirnas.ribosomals");
        addRibFileToHashmap(ribFile);
//
//                                          File groupFile = new File
(folderobjects+"\\tailles.afterPrediction\\lib_all.afterPrediction.com.uni.groupslist");
//        addGroupFileToHashmap(groupFile);

        File targetFile = new File (folder+"targetgenes/tapir_results.summary.txt");
        addTargetFileToHashmap(targetFile);

        //Conserved in mirbase and pmrd, Obtained after a blast and checked as true in checkBlats
or ConservedAgainstMirnasDBItol
        File conservedFile = new File (folder+"/blasts/all_lib.mirnas.blastmirbase.true.txt");
        addConservedFileToHashmap(conservedFile);
        conservedFile = new File (folder+"/blasts/all_lib.mirnas.blastPMRD.true.txt");
        addConservedFileToHashmap(conservedFile);

        //Found in non codants rnas
        File nonCodantFile = new File (folder+"/blasts/all_lib.mirnas.blastncrna.true.txt");
        addNonCodantsFileToHashmap(nonCodantFile);

        //Low complexity by repeatMasker
        File LowComplexityFile = new File
(folder+"/repeatMasker_results/all_lib.mirnas.RepeatMasker.txt");
        addLowComplexityFileToHashmap(LowComplexityFile);

        //mirnaStars
        File mirnastarsFile = new File (folderobjects+"mirnasStars.txt");
        addmirnastarsFileToHashmap(mirnastarsFile);
//
//        //File obtained after blast of mirnasStars against cutadapt files
//        File mirnastarsInCutadaptFile = new File (folderobjects+"mirnasStarsInCutadapt.txt");
//        mirnastarsInCutadaptFileToHashmap(mirnastarsInCutadaptFile);

        //families
        File familiesFile = new File
(folder+"/families/all_lib.pred.analyses.com.uni.mirnas.fasta.tree.group");
        addfamiliesFileToHashmap(familiesFile);

        //Quality reads, run b_mapping.mappingQuality.java
        File qualityreadsFile = new File (folderobjects+"qualityreads.txt");
        addqualityreadsFileToHashmap(qualityreadsFile);

        //awk '{print $11"\t"$16}' FS="\t" all_lib.pred.analyses.com >
all_lib.pred.analyses.com.uni.mirnaPrec (mirna+prec)
        File mirnaPrec = new File (folderobjects+"all_lib.pred.analyses.com.uni.mirnaPrec");
        addmirnaPrecFileToHashmap(mirnaPrec);

        //awk '{print $16}' FS="\t" all_lib.com | perl -ne '$H{$_}++ or print' | awk '{print
">"$1"\n"$1}'> forPrecursorsCorrectionMFE.fasta
        File correctedMFEFile = new File
(folderobjects+"forPrecursorsCorrectionMFE.fasta.folded");
        addcorrectedMFEFileToHashmap(correctedMFEFile);

        // awk '{print $11"\t"$3}' FS="\t" all_lib.pred.analyses.com | perl -ne '$H{$_}++ or
print'> all_lib.pred.analyses.com.uni.exp
        File correctedExpressionFile = new
File(folderobjects+"all_lib.pred.analyses.com.uni.exp");
        addcorrectedExpressionFileToHashmap(correctedExpressionFile);
//
        File GORetrieverFile = new
File("/ibrixfs1/Data/mik/cutadapt/targetgenes/tapir_results_genes,goretriver,clsFormat.txt");
```

```
        addGORetriverFileToHashmap(GORetrieverFile);

//        //NOT necessary, because not revelant
//                                                    File       AgriGOFile       =       new
File("H:\\mik\\_version3\\v_targetgenes\\tapir_results_unirefs.AgriGo.results");
//        addAgriGOFileToHashmap(AgriGOFile);


        addResultsToObjects();

        //  rm afterFilters_analyses/all_lib.pred.analyses.com
        // for i in {15..30};do for j in {1..10};do cat $j/lib${j}_${i}.pred.filters_analyses.com
;done;done | perl -ne '$H{$_}++ or print'>> afterFilters_analyses/all_lib.pred.analyses.com
        // OU
        // create.AllLibs.analysesCom.sh
        // Recuperer mirnas : awk '{print $11}' FS="\t" all_lib.pred.analyses.com | perl -ne
'$H{$_}++ or print' > all_lib.pred.analyses.mirnas.txt
        //    Donnees    diverses    :    cat    all_lib.pred.analyses.com    |    awk    '{print
$1"\t"$2"\t"$3"\t"$5"\t"$6"\t"$8"\t"$9"\t"$12"\t"$19"\t"$20"\t"$21"\t"$23"\t"$26}' FS="\t" | perl
-ne '$H{$_}++ or print' > all_lib.pred.analyses.com.uni.awk
    }

    /**
     *
     */
    public static void addResultsToObjects(){
        System.out.println("Compute objects");
        for (int i = 1; i <=10; i++) {
            for (int j = 15; j <= 30; j++) {
                System.out.println("lib"+i+"_"+j);
                File INobject= new File(folderobjects+i+"/lib"+i+"_"+j+".pred.obj");
                File OUTobject= new File(folderobjects+i+"/lib"+i+"_"+j+".pred.analyses.obj");

                //File                                    outExp=                                    new
File(folderobjects+i+"\\lib"+i+"_"+j+".pred.filters_analyses.exp");
                File outCom= new File(folderobjects+i+"/lib"+i+"_"+j+".pred.analyses.com");
                File outFile= new File(folderobjects+i+"/lib"+i+"_"+j+".pred.analyses");
                if (OUTobject.exists()) OUTobject.delete();
                if (outCom.exists()) outCom.delete();
                if (outFile.exists()) outFile.delete();

                int id = 0;
                int notarget=0;
                int ribosomal=0;

                try {
                    rnaobject2 r1 = new rnaobject2();
                    rnaobject2 r2 = new rnaobject2();
                    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(INobject));
                    ObjectOutputStream       oos       =       new       ObjectOutputStream(new
FileOutputStream(OUTobject, true));

                    //PrintWriter    pwexp    =    new    PrintWriter(new    BufferedWriter(new
FileWriter(outExp, true)));
                    PrintWriter pwcom = new PrintWriter(new BufferedWriter(new FileWriter(outCom,
true)));
                    PrintWriter    pwout    =    new    PrintWriter(new    BufferedWriter(new
FileWriter(outFile, true)));
                    pwcom.println(r1.toStringCommonsHeader());
                    pwout.println(r1.toStringHeader());

                    boolean b=true;
                    r1 = new rnaobject2();
                    r2 = new rnaobject2();
                    r1.setId(1);
                    r2.setId(1);
                    boolean r=true;
                    r1 = (rnaobject2) ois.readObject();
                    while (b) {
                        //teste fin du fichier pour éviter une exception
                        if (r1.getId()==0||r2.getId()==0){
                            b = false;
                        } else {
                            if (r){
                                r2 = (rnaobject2) ois.readObject();
                                r=false;
```

```
                    } else {
                        r1 = (rnaobject2) ois.readObject();
                        r=true;
                    }
                    try {
                        r1 = setting(r1);
                        r2 = setting(r2);

                        r1.setMiRNA(true);
                        r2.setMiRNA(true);
                    } catch (Exception e) {
                        //Exception souvent lorsque le miRNA n'est pas trouvé
                        //dans le précurseur lors du Set arm dans setting
                        r1.setMiRNA(false);
                        r2.setMiRNA(false);
                    }


                    // Print common only if previous object have the same id than current
                    if (r1.getId() == r2.getId()) {
                        r1.setPredictorsCommon(true);
                        r2.setPredictorsCommon(true);
                        oos.writeObject(r1);
                        oos.writeObject(r2);

                        pwout.println(r1.toString());
                        pwout.println(r2.toString());

                        //print under restrictions
                        if (commonsRestrictions(r1)&&commonsRestrictions(r2)) {
                            pwcom.println(r1.toStringCommons());
                            pwcom.println(r2.toStringCommons());
                        } else {
                            if (r1.isRibosomal()) {
                                ribosomal++;
                            } else {
                                notarget++;
                            }
                        }

                        //print all
                    } else if (r1.isMiRNA() && r2.isMiRNA()) {
                        r1.setPredictorsCommon(false);
                        r2.setPredictorsCommon(false);

                        oos.writeObject(r1);
                        oos.writeObject(r2);

                        pwout.println(r1.toString());
                        pwout.println(r2.toString());
                    }
                    id = r1.getId();

                }
            }
            //add last object at id=0 to avoid exception at reading
            r1 = new rnaobject2();
            r1.setId(0);
            oos.writeObject(r1);

            //closing
            ois.close();
            pwcom.close();
            pwout.close();
            oos.close();
            //System.out.println("In      commons      :      Ribosomaux="+ribosomal+"\tNo
targets="+notarget);
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("id = "+id);


        }
        removeDuplicates(outCom);
        removeDuplicates(outFile);
    }
```

```
        }
        concatenateFiles();
    }

    public static boolean commonsRestrictions(rnaobject2 r){
        if (!r.isMiRNA()){
            return false;
        }
        if (r.getMapper().equals("dump")){
            return false;
        }
        if (r.isConserved()){
            return true;
        }
        if (r.isRibosomal()){
            return false;
        }
        if (r.isNonCodant()){
            return false;
        }
        if (r.getTargetGenesAmountFromList()==0){
            return false;
        }
        if (r.isLowComplexity()){
            return false;
        }
        if (r.getMappingMismatchs()>=3){
            return false;
        }
        if (r.getMappingMismatchsCorrected()>=3){
            return false;
        } else return true;
    }

    private static rnaobject2 setting(rnaobject2 r) {
        //newr = newr.convertRnaObject(r);

        //set mappingSourceName
        String mappingAcc=r.getMappingSourceAccession();
        String mappingSN=hmrefs.get(mappingAcc);
        r.setMappingSourceName(mappingSN);


        String rna = r.getSmallRNAsequenceAfterMappingWithNT();

        // set ribosomaux
        if (hmrib.get(rna)==null){
            r.setRibosomal(false);
        } else {
            r.setRibosomal(true);
            r.setRibosomalName(hmrib.get(rna));
        }

//        if (!r.getMappingSourceName().contains("ribosom")){
//            r.setRibosomal(false);
//        } else {
//            r.setRibosomal(true);
//            r.setRibosomalName(r.getMappingSourceName());
//        }


        //Set mirnaStar and its abondance
        String mirnaStar = hmstars.get(rna+"/"+r.getPrecursorSequence());
        r.setMirnaStar(mirnaStar);
        if (hmstarsCutadapt.containsKey(mirnaStar)){
            r.setMirnaStarAbondance(hmstarsCutadapt.get(mirnaStar));
        } else r.setMirnaStarAbondance(0);


        //set target genes
        r.setTargetgenesList(hmtargets.get(rna));

        //Set Unirefs
        String unirefs="";
```

```
        if (hmtargets.get(rna)!=null){
            String tabs[]=hmtargets.get(rna).split(";");
            HashMap<String,Double> hm = new HashMap<String,Double>();
            for (String tab : tabs) {
                if(tab.contains("UniRef")){
                    int idx = tab.indexOf("UniRef")+10;
                    double scor=Double.valueOf(tab.split(",")[3]);
                    tab=tab.substring(idx);
                    String uniref=tab.substring(0, tab.indexOf(" "));
                    if (!hm.containsKey(uniref)){
                        hm.put(uniref,scor);
                    }
                }
            }
            for (String uni : hm.keySet()) {
                unirefs=unirefs+uni+";";
            }

            if (unirefs.length()>0){
                unirefs=unirefs.substring(0, unirefs.length()-1);
                r.setUnirefsfromTargetGenes(unirefs);
            }
        }

        // Set groups
        String exp = r.getExpressionPerLibToStringVirgules()+","+r.getExpressionTotal();
        r.setGroupIDafterPrediction(hmgroups.get(exp));

        // Set conserved
        if (hmconserved.get(rna)!=null){
            r.setConserved(true);
        }

        // Set nonCodant
        if (hmNonCodants.get(rna)!=null){
            r.setNonCodant(true);
        }
        // Set low complexity
        if (hmLC.get(rna)!=null){
            r.setLowComplexity(true);
        }

        //Set category
        if (r.getPredictorName().equals("mipred")){
            if
(Double.valueOf(r.getPredictorScore())<0.05&&r.getMipredRealpseudo().equals("real")){
                r.setCategory('A');
            }
            if
(Double.valueOf(r.getPredictorScore())<0.05&&r.getMipredRealpseudo().equals("pseudo")){
                r.setCategory('B');
            }
            if
(Double.valueOf(r.getPredictorScore())>=0.05&&r.getMipredRealpseudo().equals("real")){
                r.setCategory('C');
            }
            if
(Double.valueOf(r.getPredictorScore())>=0.05&&r.getMipredRealpseudo().equals("pseudo")){
                r.setCategory('D');
            }
        } else r.setCategory('N');

         //set family
        if (hmfamily.containsKey(rna)){
            r.setFamily("f"+hmfamily.get(rna));
        }

         //set qualityValue and lib
        String read = r.getSmallRNAcolorName();
        if (hmreadqual.containsKey(read)){
            int occ = Integer.valueOf(hmreadqual.get(read).split("\t")[0]);
            String lib = hmreadqual.get(read).split("\t")[1];
            r.setOriginalColorLibrary(lib);
            r.setOriginalQualityRead(occ);
        }
```

```
        //Set number of different precursors
        if (hmmirnaPrec.containsKey(rna)){
            r.setNumberOfDistinctPrecursors(hmmirnaPrec.get(rna));
        }

        //Set corrected MFE precursors
        String prec = r.getPrecursorSequence();
        if (hmMFEPrec.containsKey(prec)){
            r.setPrecursorStructureMFEcorrected(hmMFEPrec.get(prec));
        }

        //Set corrected expression profiles
        if (hmexp.containsKey(rna)){
            r.setExpressionPerLibCorrected(hmexp.get(rna));
        } else {
            r.setExpressionPerLibCorrected(r.getExpressionPerLib());
        }

        //Set Gene Ontologies
        String p="";
        String f="";
        String c="";
        String ProtName="";
        String ProtID="";
        if (unirefs.length()>1) {
            String unir[] = r.getUnirefsfromTargetGenes().split(";");

            for (String uniref : unir) {
                uniref=uniref.trim();
                p=p+";"+hmgoP.get(uniref);
                f=f+";"+hmgoF.get(uniref);
                c=c+";"+hmgoC.get(uniref);
                ProtName=ProtName+";"+hmgoProtName.get(uniref);
                ProtID=ProtID+";"+hmgoProtID.get(uniref);

            }
            r.setGOProcessBiologic(p.substring(1).replace("null;", ""));
            r.setGOFunctionMolecular(f.substring(1).replace("null;", ""));
            r.setGOComponentCell(c.substring(1).replace("null;", ""));
            r.setGOproteinName(ProtName.substring(1).replace("null;", ""));
            r.setGOproteinID(ProtID.substring(1).replace("null;", ""));
        }

        //Set arm
        String arm = getArm(r.getSmallRNAsequenceAfterMapping(), r.getPrecursorSequence(),
r.getPrecursorStructure());
        r.setArmOfSmallRNAOnPrecursor(arm);

        return r;
    }

    private static void addRibFileToHashmap(File ribFile) {
        int countlines = tools.countLines(ribFile);
        System.out.println("Adding Ribosomes to hashmap");
        hmrib = new HashMap<String, String>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(ribFile));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                String mirna = line.split("\t")[0].replaceAll("T", "U");
                String rib = line.split("\t")[1];
                if (hmrib.get(mirna)!=null){
                    //System.err.println("COLLISION at "+mirna.replaceAll("U", "T"));
                }
                hmrib.put(mirna, rib);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in addRibosomesToHashmap at "+i+" on "+countlines);
        }
    }
```

```
private static void addGroupFileToHashmap(File groupFile) {
    int countlines = tools.countLines(groupFile);
    System.out.println("Adding Groups to hashmap");
    hmgroups = new HashMap<String, String>(countlines);
    int i=0;
    try {
        BufferedReader br = new BufferedReader(new FileReader(groupFile));
        String line="";
        while (br.ready()){
            i++;
            line = br.readLine();
            String exp = line.split("\t")[2];
            String group = line.split("\t")[0];
            if (hmgroups.get(exp)!=null&&!group.equals(hmgroups.get(exp))){
                System.err.println("COLLISION at "+exp);
            }
            hmgroups.put(exp, group);
        }
        br.close();
    } catch (Exception ex) {
        ex.printStackTrace();
        System.err.println("Error in addGroupsToHashmap at "+i+" on "+countlines);
    }
}

/**
 * Add target file to hashmap
 * @param targetFile
 */
private static void addTargetFileToHashmap(File targetFile) {
    int countlines = tools.countLines(targetFile);
    System.out.println("Adding Targets to hashmap");
    hmtargets = new HashMap<String, String>(countlines);
    int i=0;
    try {
        BufferedReader br = new BufferedReader(new FileReader(targetFile));
        String line="";
        while (br.ready()){
            i++;
            line = br.readLine();
            String mirna = line.split("\t")[0].replaceAll("T", "U");
            String targets = line.split("\t")[1];
            if (hmtargets.get(mirna)!=null){
                System.err.println("COLLISION at "+mirna);
            }
            hmtargets.put(mirna, targets);
        }
        br.close();

    } catch (Exception ex) {
        ex.printStackTrace();
        System.err.println("Error in addTargetsToHashmap at "+i+" on "+countlines);
    }
}

/**
 * Add conserved file to hashmap
 * @param conservedFile
 */
private static void addConservedFileToHashmap(File conservedFile) {
    int countlines = tools.countLines(conservedFile);
    System.out.println("Adding Conserved to hashmap");
    //hmconserved = new HashMap<String, String>(countlines);
    int i=0;
    try {
        BufferedReader br = new BufferedReader(new FileReader(conservedFile));
        String line="";
        line = br.readLine();
        while (br.ready()){
            i++;
            line = br.readLine();
            String l[]=line.split("\t");
            String mirna=l[0];
            String family= l[4];
            hmconserved.put(mirna, family);
        }
```

```
            br.close();
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in addConservedFileToHashmap at "+i+" on "+countlines);
        }
    }

    /**
     * Add non codant file to hashmap
     * @param nonCodant File
     */
    private static void addNonCodantsFileToHashmap(File nonCodantFile) {
        int countlines = tools.countLines(nonCodantFile);
        System.out.println("Adding non Codants to hashmap");
        hmNonCodants = new HashMap<String, String>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(nonCodantFile));
            String line="";
            line = br.readLine();
            while (br.ready()){
                i++;
                line = br.readLine();
                String l[]=line.split("\t");
                String mirna=l[0];
                String type;
                try {
                    type = l[2].substring(l[2].lastIndexOf("_"));
                } catch (Exception e) {
                    type = l[2];
                }
                hmNonCodants.put(mirna, type);
            }
            br.close();
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in addNonCodantsFileToHashmap at "+i+" on "+countlines);
        }
    }

     /**
     * Add LowComplexity file to hashmap
     * @param LowComplexityFile
     */
    private static void addLowComplexityFileToHashmap(File LowComplexityFile) {
        int countlines = tools.countLines(LowComplexityFile);
        System.out.println("Adding LowComplexityFile to hashmap");
        hmLC = new HashMap<String, String>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(LowComplexityFile));
            String line="";
            line = br.readLine();
            while (br.ready()){
                i++;
                line = br.readLine();
                String l[]=line.split(" ");
                String mirna=l[4];
                String type=l[8];
                hmLC.put(mirna, type);
            }
            br.close();
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in addLowComplexityFileToHashmap at "+i+" on "+countlines);
        }
    }


    private static String getArm(String smallRNAsequenceAfterMapping, String precursorSequence,
String precursorStructure) {
        int mirnaStart=precursorSequence.indexOf(smallRNAsequenceAfterMapping);
        int mirnaEnd=mirnaStart+smallRNAsequenceAfterMapping.length();
        if (precursorStructure.substring(mirnaStart, mirnaEnd).contains("(")){
            return "5'";
        } else return "3'";
```

```
    }

    private static void addmirnastarsFileToHashmap(File mirnastarsFile) {
        int countlines = tools.countLines(mirnastarsFile);
        System.out.println("Adding mirnaStars to hashmap");
        hmstars = new HashMap<String, String>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(mirnastarsFile));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                String tab[]=line.split("\t");
                //key = mirna/precurseur
                // pour éviter les doublons
                String key = tab[0]+"/"+tab[1];
                String mirnastar = tab[3];
                int taille = Integer.valueOf(tab[4]);
                if (!mirnastar.contains("loop")||!mirnastar.contains("error")||taille>=7){
                    hmstars.put(key, mirnastar);
                }
            }
            br.close();
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in addmirnasStarsFileToHashmap at "+i+" on "+countlines);
        }
    }

    private static void mirnastarsInCutadaptFileToHashmap(File mirnastarsInCutadaptFile) {
        int countlines = tools.countLines(mirnastarsInCutadaptFile);
        System.out.println("Adding mirnastarsInCutadapt to hashmap");
        hmstarsCutadapt = new HashMap<String, Integer>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(mirnastarsInCutadaptFile));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                String tab[]=line.split("\t");
                String mirnastar = tab[0];
                int abondance = Integer.valueOf(tab[1]);
                hmstarsCutadapt.put(mirnastar.replaceAll("T", "U"),abondance);
            }
            br.close();
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in mirnastarsInCutadaptFileToHashmap at "+i+" on
"+countlines);
        }
    }

    private static void addfamiliesFileToHashmap(File familiesFile) {
        int countlines = tools.countLines(familiesFile);
        System.out.println("Adding familiesFile to hashmap");
        hmfamily = new HashMap<String, Integer>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(familiesFile));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                String tab[]=line.split("\t");
                String mirna = tab[0];
                int family = Integer.valueOf(tab[1]);
//                if (mirna.contains("miR")){
//                    String t[]=mirna.split("-");
//                    mirna=t[1].substring(mirna.indexOf("miR")+3);
//                    try {
//
//                    } catch (Exception e) {
//                    }
//                }
```

```
            hmfamily.put(mirna,family);
        }
        br.close();
    } catch (Exception ex) {
        ex.printStackTrace();
        System.err.println("Error in familiesFileToHashmap at "+i+" on "+countlines);
    }
}

private static void addqualityreadsFileToHashmap(File qualityreadsFile) {
    int countlines = tools.countLines(qualityreadsFile);
    System.out.println("Adding qualityreadsFile to hashmap");
    hmreadqual = new HashMap<String, String>(countlines);
    int i=0;
    try {
        BufferedReader br = new BufferedReader(new FileReader(qualityreadsFile));
        String line="";
        while (br.ready()){
            i++;
            line = br.readLine();
            String tab[]=line.split("\t");
            String read = tab[0];
            String occurences = tab[1];
            String lib = tab[2];
            hmreadqual.put(read,occurences+"\t"+lib);
        }
        br.close();
    } catch (Exception ex) {
        ex.printStackTrace();
        System.err.println("Error in qualityreadsFileToHashmap at "+i+" on "+countlines);
    }
}

private static void addmirnaPrecFileToHashmap(File mirnaPrec) {
    int countlines = tools.countLines(mirnaPrec);
    System.out.println("Adding mirnaPrecFile to hashmap");
    hmmirnaPrec = new HashMap<String, Integer>(countlines);
    int i=0;
    try {
        BufferedReader br = new BufferedReader(new FileReader(mirnaPrec));
        String line="";
        while (br.ready()){
            i++;
            line = br.readLine();
            String tab[]=line.split("\t");
            String mirna = tab[0];
            //String prec = tab[1];
            if (hmmirnaPrec.containsKey(mirna)){
                int tmp=hmmirnaPrec.get(mirna);
                tmp++;
                hmmirnaPrec.put(mirna, tmp);
            } else {
                hmmirnaPrec.put(mirna, 1);
            }
        }
        br.close();
    } catch (Exception ex) {
        ex.printStackTrace();
        System.err.println("Error in mirnaPrecFileToHashmap at "+i+" on "+countlines);
    }
}

/**
 * Add corrected MFEs
 * @param correctedMFEFile
 */
private static void addcorrectedMFEFileToHashmap(File correctedMFEFile) {
    int countlines = tools.check_number_contigs(correctedMFEFile);
    System.out.println("Adding correctedMFEFile to hashmap");
    hmMFEPrec = new HashMap<String, Double>(countlines);
    int i=0;
    try {
        BufferedReader br = new BufferedReader(new FileReader(correctedMFEFile));
        String line="";
        while (br.ready()){
            i++;
```

```
                line = br.readLine();
                if (line.startsWith(">")){
                    String seq=br.readLine();
                    line=br.readLine();
                    Double mfe=Double.valueOf(line.substring(line.indexOf(" ")+2,line.length()-
1));
                    hmMFEPrec.put(seq, mfe);
                }
            }
            br.close();
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in mirnaPrecFileToHashmap at "+i+" on "+countlines);
        }
    }

    private static void addcorrectedExpressionFileToHashmap(File correctedExpressionFile) {
        int countlines = tools.countLines(correctedExpressionFile);
        System.out.println("Adding correctedExpressionFile to hashmap");
        hmexp = new HashMap<String, int[]>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(correctedExpressionFile));
            String line="";
            while (br.ready()){
                i++;
                line = br.readLine();
                String t[]=line.split("\t");
                String seq = t[0];

                String tab[]=t[1].split(",");
                int exp[]=new int[10];
                for (int j = 0; j < tab.length; j++) {
                    exp[j] = Integer.valueOf(tab[j]);
                }
                if (hmexp.containsKey(seq)){
                    int tmp[]=hmexp.get(seq);
                    for (int j = 0; j < tmp.length; j++) {
                        tmp[j]=tmp[j]+exp[j];
                    }
                    hmexp.put(seq,tmp);
                } else {
                    hmexp.put(seq,exp);
                }
            }
            br.close();
        } catch (Exception ex) {
            ex.printStackTrace();
            System.err.println("Error in correctedExpressionFileToHashmap at "+i+" on
"+countlines);
        }
    }

    private static void addGORetriverFileToHashmap(File GOFile) {
        int countlines = tools.countLines(GOFile);
        System.out.println("Adding GOFileToHashmap to hashmap");
        hmgoP = new HashMap<String, String>(13000);
        hmgoC = new HashMap<String, String>(13000);
        hmgoF = new HashMap<String, String>(13000);
        hmgoProtName = new HashMap<String, String>(countlines);
        hmgoProtID = new HashMap<String, String>(countlines);
        int i=0;
        try {
            BufferedReader br = new BufferedReader(new FileReader(GOFile));
            String line=br.readLine();
            while (br.ready()){
                i++;
                line = br.readLine();
                String tab[]=line.split("\t");
                String term_type = tab[10];
                String proteinID = tab[3];
                String proteinName = tab[11];
                String term = tab[6];

                String unirefs[] = {tab[0]};
                for (String uniref : unirefs) {
```

```
                uniref=uniref.trim();
                if (uniref.length()>1) {
                    if (term_type.equals("P")) {
                        if (hmgoP.containsKey(uniref)) {
                            String tmp = hmgoP.get(uniref);
                            if (!tmp.contains(term)){
                                tmp = tmp + ";" + term;
                                hmgoP.put(uniref, tmp);
                            }
                        } else {
                            hmgoP.put(uniref, term);
                        }
                    }
                    if (term_type.equals("F")) {
                        if (hmgoF.containsKey(uniref)) {
                            String tmp = hmgoF.get(uniref);
                            if (!tmp.contains(term)){
                                tmp = tmp + ";" + term;
                                hmgoF.put(uniref, tmp);
                            }
                        } else {
                            hmgoF.put(uniref, term);
                        }
                    }
                    if (term_type.equals("C")) {
                        if (hmgoC.containsKey(uniref)) {
                            String tmp = hmgoC.get(uniref);
                            if (!tmp.contains(term)){
                                tmp = tmp + ";" + term;
                                hmgoC.put(uniref, tmp);
                            }
                        } else {
                            hmgoC.put(uniref, term);
                        }
                    }
                }
                hmgoProtID.put(uniref,proteinID);
                hmgoProtName.put(uniref,proteinName);
            }
        }

        br.close();
    } catch (Exception ex) {
        ex.printStackTrace();
        System.err.println("Error in GOFileToHashmap at "+i+" on "+countlines);
    }
}

    private static void addAgriGOFileToHashmap(File GOFile) {
    int countlines = tools.countLines(GOFile);
    System.out.println("Adding GOFileToHashmap to hashmap");
    hmgoP = new HashMap<String, String>(13000);
    hmgoC = new HashMap<String, String>(13000);
    hmgoF = new HashMap<String, String>(13000);
    hmgoProtName = new HashMap<String, String>(countlines);
    int i=0;
    try {
        BufferedReader br = new BufferedReader(new FileReader(GOFile));
        String line=br.readLine();
        while (br.ready()){
            i++;
            line = br.readLine();
            String tab[]=line.split("\t");
            String term_type = tab[1];
            String term = tab[2];
            String pvalue = tab[7];
            String entries[] = tab[9].split("//");
            for (String entry : entries) {
                entry=entry.trim();
                if (entry.length()>1) {
                    if (term_type.equals("P")) {
                        if (hmgoP.containsKey(entry)) {
                            String tmp = hmgoP.get(entry);
                            if (!tmp.contains(term)){
                                tmp = tmp + ";" + term;
                                hmgoP.put(entry, tmp);
```

```
                        }
                    } else {
                        hmgoP.put(entry, term);
                    }
                }
                if (term_type.equals("F")) {
                    if (hmgoF.containsKey(entry)) {
                        String tmp = hmgoF.get(entry);
                        if (!tmp.contains(term)){
                            tmp = tmp + ";" + term;
                            hmgoF.put(entry, tmp);
                        }
                    } else {
                        hmgoF.put(entry, term);
                    }
                }
                if (term_type.equals("C")) {
                    if (hmgoC.containsKey(entry)) {
                        String tmp = hmgoC.get(entry);
                        if (!tmp.contains(term)){
                            tmp = tmp + ";" + term;
                            hmgoC.put(entry, tmp);
                        }
                    } else {
                        hmgoC.put(entry, term);
                    }
                }
            )
            hmgoProtName.put(entry,pvalue);
        }
    )

    br.close();
} catch (Exception ex) {
    ex.printStackTrace();
    System.err.println("Error in GOFileToHashmap at "+i+" on "+countlines);
)
}

private static void addRefsToHashMap(File refsFile) {
    int countlines = tools.countLines(refsFile);
    System.out.println("Adding refsFile to hashmap");
    hmreadqual = new HashMap<String, String>(countlines);
    int i=0;
    try {
        BufferedReader br = new BufferedReader(new FileReader(refsFile));
        String line="";
        while (br.ready()){
            i++;
            line = br.readLine();
            String name = null;
            String acc = null;
            try {
                int ind = line.indexOf(" ");
                acc = line.substring(0, ind);
                name = line.substring(ind + 1);
                name = name.replace(":", "_").replace(",", "_").replace(";", "_");
            } catch (Exception e) {
                name="Unknown";
            }

            hmrefs.put(acc,name);
        }
        br.close();
    } catch (Exception ex) {
        ex.printStackTrace();
        System.err.println("Error in refsFileToHashmap at "+i+" on "+countlines);
    }
}

private static void removeDuplicates(File outfile) {
    File tmpFile=new File(outfile+".tmp");
    tools.executeLinuxCommand("perl -ne '$H{$_}++ or print' "+outfile+" > "+tmpFile);
    outfile.delete();
    tmpFile.renameTo(outfile);
}
```

```
    private static void concatenateFiles() {
        tools.executeLinuxCommand("sh
/ibrixfsl/Data/mik/cutadapt/objects/create.AllLibs.analysesCom.sh");
        //tools.executeLinuxCommand("cat      all_lib.pred.analyses.com      |      awk      '{print
$11\"\t\"$1\"\t\"$2\"\t\"$5\"\t\"$6\"\t\"$7\"\t\"$8\"\t\"$9\"\t\"$12\"\t\"$13\"\t\"$14\"\t\"$16\"
\t\"$18\"\t\"$20\"\t\"$21\"\t\"$22\"\t\"$23\"\t\"$24\"\t\"$25\"\t\"$26\"\t\"$27}'    FS=\"\t\"    |
perl -ne '$H{$_}++ or print' > all_lib.pred.analyses.com.uni.awk");
    }
}
```

rnaobject2.java

rnaobject2 (2 car il y avait une version précédente mais oubliée) est l'object java en tant que tel, où toutes les caractéristiques des microARNs prédits sont déterminées.

```
/*
 * rnaobject version 2, avec les paramétres qui manquaient
 */

package tools;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.List;

/**
 *
 * @author Mickael
 */
public class rnaobject2 implements Serializable {

    static final long serialVersionUID = 7526472295622776147L;

    /**
     * id donné
     */
    private  int id;
    /**
     * nom du code couleur
     */
    private  String smallRNAcolorName;
    /**
     * tableau d'expression, une case par librairie
     */
    private  int[] expressionPerLib;
    /**
     * tableau d'expression corrigé, une case par librairie
     * Si plusieurs mêmes miRNAs prédits ont un même profil d'expression,
     * ils sont alors additionnés, et donc corrigés
     */
    private  int[] expressionPerLibCorrected;
    /**
     * expression au total
     */
    private  int expressionTotal;
    private  int expressionTotalCorrected;
    /**
     * sequence originale avant le mapping
     */
    private  String originalSmallRNAsequence;
    /**
     * Librairie originale de la sequence couleur
     */
    private  String originalColorLibrary;
    /**
     * Nombre de base ayant une quality value under 10
     */
    private  int originalQualityRead;


    /**
     * nom du mapper
```

```
 */
private  String mapper;
/**
 * la source corresond à l'EST
 */
private  String mappingSourceAccession;
/**
 * nom de l'EST
 */
private  String mappingSourceName;
/**
 * sens sur l'EST
 */
private  String strandOnSource;
/**
 * position de depart sur l'EST
 */
private  int startPositionOnSource;
/**
 * mismatchs lors du mapping contre l'EST
 */
private  int mappingMismatchs;
private  int mappingMismatchsCorrected;
private  boolean mappingMismatchsAdded;
/**
 * Sequence du smallRNA sur l'EST,apres mapping
 */
private  String smallRNAsequenceAfterMapping;
private  String smallRNAsequenceAfterMappingWithNT;
private  String smallRNAsequenceAfterMappingWithNTdone;

/**
 * Number of distinct precursors of the smallRNA sequence after mapping
 */
private  int numberOfDistinctPrecursors;
/**
 * Sequence du smallRNA star sur l'EST,apres mapping
 */
private  String mirnaStar;
/**
 * Sequence du smallRNA star sur l'EST,apres mapping
 */
private  int mirnaStarAbondance;

/**
 * Sur quelle branche du precurseur se trouve le smallR
 */
private  String armOfSmallRNAOnPrecursor;



/**
 * sequence du precurseur
 */
private  String precursorSequence;
/**
 * structure secondaire du precurseur provenant de RNAfold
 */
private  String precursorStructure;
/**
 * MFE du folding
 */
private  double precursorStructureMFE;

private  double precursorStructureMFEcorrected;

/**
 * nom du predicteur
 */
private  String predictorName;
/**
 *
 */
private boolean predictorsCommon;
/**
 * score du predicteur
```

```
 */
private  String predictorScore;
private  String mipredRealpseudo;
private  double mipredConfidence;

/**
 * si miRNA ou non
 */
private  boolean miRNA;
/**
 * Category
 */
private  char category;

/**
 * si detecté comme ribosomal ou non
 */
private  boolean ribosomal;
/**
 * nom du ribosome
 */
private  String ribosomalName;
/**
 * filtre applique en plus
 */
private  String filter;

/**
 * Conserved
 */
private boolean conserved;

/**
 * Non Codant
 */
private boolean nonCodant;
/**
 * Non Codant
 */
private boolean lowComplexity;

/**
 * id du groupe si on fait des groupes
 */
private  String groupIDafterPrediction;
/**
 * nom du groupe si besoin
 */
private  String groupNameafterPrediction;
/**
 * nom de la famille
 */
private  String Family;

//target genes, organisés en vecteurs

/**
 * targetgenes with accession,name,start,score separated by ;
 */
private String targetgenesList;
private  String unirefsfromTargetGenes; //
private  String GOProcessBiologic; //
private  String GOComponentCell; //
private  String GOFunctionMolecular; //
private  String GOproteinName; //
private  String GOproteinID; //

private  String notes;  //autres commentaires


public  void rnaObject2(int id){

}

/**
```

```
 * Convertit l'objet rnaobject en rnaobject2
 * @param r
 * @return
 */
public rnaobject2 convertRnaObject(rnaobject r){
    rnaobject2 newr = this;

    this.setId(r.getId());
    this.setSmallRNAcolorName(r.getSmallRNAcolorName());
    this.setExpressionPerLib(r.getExpressionPerLib());
    this.setExpressionTotal(r.getExpressionTotal());
    this.setOriginalSmallRNAsequence(r.getOriginalSmallRNAsequence());

    this.setMapper(r.getMapper());
    this.setMappingSourceAccession(r.getMappingSourceAccession());
    this.setMappingSourceName(r.getMappingSourceName());
    this.setStrandOnSource(r.getStrandOnSource());
    this.setStartPositionOnSource(r.getStartPositionOnSource());
    this.setMappingMismatchs(r.getMappingMismatchs());
    this.setSmallRNAsequenceAfterMapping(r.getSmallRNAsequenceAfterMapping());

    this.setPrecursorSequence(r.getPrecursorSequence());
    this.setPrecursorStructure(r.getPrecursorStructure());
    this.setPrecursorStructureMFE(r.getPrecursorStructureMFE());

    this.setPredictorName(r.getPredictorName());
    this.setPredictorScore(r.getPredictorScore());

    this.setMiRNA(r.isMiRNA());

    this.setNotes(r.getNotes());

    return newr;
}


/**
 * @return the id
 */
public  int getId() {
    return id;
}

/**
 * @param aId the id to set
 */
public  void setId(int aId) {
    id = aId;
}

/**
 * @return the smallRNAcolorName
 */
public  String getSmallRNAcolorName() {
    return smallRNAcolorName;
}

/**
 * @param aSmallRNAcolorName the smallRNAcolorName to set
 */
public  void setSmallRNAcolorName(String aSmallRNAcolorName) {
    smallRNAcolorName = aSmallRNAcolorName;
}


/**
 * @return the expressionPerLib
 */
public  int[] getExpressionPerLib() {
    return expressionPerLib;
}

/**
 * @return the expressionPerLib 1,2,3,4,5,6,7,8,9,10
 */
public String getExpressionPerLibToStringVirgules() {
```

```
    if (this.getExpressionPerLib()!=null){
        String s = "";
        for (int i = 0; i <= 9; i++) {
            if (i!=9){
                s+=this.getExpressionPerLib()[i]+",";
            }else {
                s+=this.getExpressionPerLib()[i];
            }
        }
        return s;
    } else {
        return "0,0,0,0,0,0,0,0,0,0";
    }
}

/**
 * @return the expressionPerLib 1,2,3,4,5,6,7,8,9,10
 */
public String getExpressionPerLibCorrectedToStringVirgules() {
    if (this.getExpressionPerLibCorrected()!=null){
        String s = "";
        for (int i = 0; i <= 9; i++) {
            if (i!=9){
                s+=this.getExpressionPerLibCorrected()[i]+",";
            }else {
                s+=this.getExpressionPerLibCorrected()[i];
            }
        }
        return s;
    } else {
        return "0,0,0,0,0,0,0,0,0,0";
    }
}

/**
 * @return the expressionPerLib 1\t2\t3\t4\t5\t6\t7\t8\t9\t10
 */
public String getExpressionPerLibToStringTabs() {
    if (this.getExpressionPerLib()!=null){
        String s = "";
        for (int i = 0; i <= 9; i++) {
            if (i!=9){
                s+=this.getExpressionPerLib()[i]+"\t";
            }else {
                s+=this.getExpressionPerLib()[i];
            }
        }
        return s;
    } else {
        return "0\t0\t0\t0\t0\t0\t0\t0\t0\t0";
    }

}

/**
 * @param aExpressionPerLib the expressionPerLib to set
 */
public  void setExpressionPerLib(int[] aExpressionPerLib) {
    expressionPerLib = aExpressionPerLib;
}

/**
 * @return the expressionTotal
 */
public  int getExpressionTotal() {
    return expressionTotal;
}

/**
 * @param aExpressionTotal the expressionTotal to set
 */
public  void setExpressionTotal(int aExpressionTotal) {
    expressionTotal = aExpressionTotal;
}
```

```
/**
 * @return the expressionTotalCorrected
 */
public int getExpressionTotalCorrected() {
    try {
        int tab[] = this.getExpressionPerLibCorrected();
        int total = 0;
        for (int i : tab) {
            total += i;
        }
        this.setExpressionTotalCorrected(total);
        return expressionTotalCorrected;
    } catch (Exception e) {
        return 0;
    }
}

/**
 * @param expressionTotalCorrected the expressionTotalCorrected to set
 */
public void setExpressionTotalCorrected(int expressionTotalCorrected) {
    this.expressionTotalCorrected = expressionTotalCorrected;
}

/**
 * @return the originalSmallRNAsequence
 */
public  String getOriginalSmallRNAsequence() {
    return originalSmallRNAsequence;
}

/**
 * @param aOriginalSmallRNAsequence the originalSmallRNAsequence to set
 */
public  void setOriginalSmallRNAsequence(String aOriginalSmallRNAsequence) {
    originalSmallRNAsequence = aOriginalSmallRNAsequence;
}

/**
 * @return the mapper
 */
public  String getMapper() {
    return mapper;
}

/**
 * @param aMapper the mapper to set
 */
public  void setMapper(String aMapper) {
    mapper = aMapper;
}

/**
 * @return the mappingSourceAccession
 */
public  String getMappingSourceAccession() {
    return mappingSourceAccession;
}

/**
 * @param aMappingSourceAccession the mappingSourceAccession to set
 */
public  void setMappingSourceAccession(String aMappingSourceAccession) {
    mappingSourceAccession = aMappingSourceAccession;
}

/**
 * @return the mappingSourceName
 */
public  String getMappingSourceName() {
    return mappingSourceName;
}

/**
 * @param aMappingSourceName the mappingSourceName to set
 */
```

```
    public  void setMappingSourceName(String aMappingSourceName) {
        mappingSourceName = aMappingSourceName;
    }

    /**
     * @return the strandOnSource
     */
    public  String getStrandOnSource() {
        return strandOnSource;
    }

    /**
     * @param aStrandOnSource the strandOnSource to set
     */
    public  void setStrandOnSource(String aStrandOnSource) {
        strandOnSource = aStrandOnSource;
    }

    /**
     * @return the startPositionOnSource
     */
    public int getStartPositionOnSource() {
        return startPositionOnSource;
    }

    /**
     * @param aStartPositionOnSource the startPositionOnSource to set
     */
    public  void setStartPositionOnSource(int aStartPositionOnSource) {
        startPositionOnSource = aStartPositionOnSource;
    }

    /**
     * @return the mappingMismatchs
     */
    public  int getMappingMismatchs() {
        return mappingMismatchs;
    }

    /**
     * @param aMappingMismatchs the mappingMismatchs to set
     */
    public  void setMappingMismatchs(int aMappingMismatchs) {
        mappingMismatchs = aMappingMismatchs;
    }

    /**
     * @return the smallRNAsequenceAfterMapping
     */
    public  String getSmallRNAsequenceAfterMapping() {
        return smallRNAsequenceAfterMapping;
    }

     /**
     * @return the smallRNAsequenceAfterMapping with missing nt took on the precursor and compare
it on color read
     */
    public  String getSmallRNAsequenceAfterMappingWithNT() {


        try {
            int l = smallRNAsequenceAfterMapping.length();
            //get nucleotide on precursor

            String prec = this.getPrecursorSequence();
            if (prec.contains("T")){
                prec=prec.replaceAll("T", "U");
            }
            int ind = prec.indexOf(smallRNAsequenceAfterMapping);
            smallRNAsequenceAfterMappingWithNT = prec.substring(ind, ind + l + 1);
            char
sequencelastNt=smallRNAsequenceAfterMapping.charAt(smallRNAsequenceAfterMapping.length()-1);
            char
correctedSequenceWithNTlastNt=smallRNAsequenceAfterMappingWithNT.charAt(smallRNAsequenceAfterMapp
ingWithNT.length()-1);
```

```
            //get real nucleotide by reading color
            char    colorLastNt=originalSmallRNAsequence.charAt(originalSmallRNAsequence.length()-
1);
            int color=4;
            if (colorLastNt=='A') color=0;
            if (colorLastNt=='C') color=1;
            if (colorLastNt=='G') color=2;
            if (colorLastNt=='T') color=3;
            char missingNT;
            if (sequencelastNt=='U'){
                missingNT=tools.hmc.get("T"+color);
            }else {
                missingNT=tools.hmc.get(sequencelastNt+""+color);
            }


            //Check if nucleotide taken on precursor is the same as the color translation
            setSmallRNAsequenceAfterMappingWithNTdone(smallRNAsequenceAfterMappingWithNT);
            if (correctedSequenceWithNTlastNt==missingNT){
                this.setMappingMismatchsAdded(false);
                mappingMismatchsCorrected=mappingMismatchs;
                return smallRNAsequenceAfterMappingWithNT;
            } else {
                mappingMismatchsCorrected=mappingMismatchs+1;
                this.setMappingMismatchsAdded(true);
                return smallRNAsequenceAfterMappingWithNT;
            }

        } catch (Exception e) {
            //e.printStackTrace();
            smallRNAsequenceAfterMappingWithNT=smallRNAsequenceAfterMapping+"-";
            setSmallRNAsequenceAfterMappingWithNTdone(smallRNAsequenceAfterMappingWithNT);
            return smallRNAsequenceAfterMappingWithNT;
        }

    }

     /**
     * @return the smallRNAsequenceAfterMappingWithNTdone
     */
    public String getSmallRNAsequenceAfterMappingWithNTdone() {
        return smallRNAsequenceAfterMappingWithNTdone;
    }

    /**
     * @param  smallRNAsequenceAfterMappingWithNTdone  the  smallRNAsequenceAfterMappingWithNTdone
to set
     */
    public                  void                  setSmallRNAsequenceAfterMappingWithNTdone(String
smallRNAsequenceAfterMappingWithNTdone) {
        this.smallRNAsequenceAfterMappingWithNTdone = smallRNAsequenceAfterMappingWithNTdone;
    }

//     /**
//     * @return the smallRNAsequenceAfterMapping with missing nt took on the precursor
//     */
//
//    public  String getSmallRNAsequenceAfterMappingWithNT2() {
//
//        try {
//            int l = smallRNAsequenceAfterMapping.length();
//            //String s = SmallRNAsequenceAfterMapping;
//            String prec = this.getPrecursorSequence();
//            int ind = prec.indexOf(smallRNAsequenceAfterMapping);
//            smallRNAsequenceAfterMappingWithNT = prec.substring(ind, ind + l + 1);
//            //System.out.println(s+"\n"+SmallRNAsequenceAfterMapping);
//        } catch (Exception e) {
//            //e.printStackTrace();
//            smallRNAsequenceAfterMappingWithNT=smallRNAsequenceAfterMapping+"-";
//        }
//        return smallRNAsequenceAfterMappingWithNT;
//    }


    /**
     * @param aSmallRNAsequenceAfterMapping the smallRNAsequenceAfterMapping to set
```

```
     */
    public  void setSmallRNAsequenceAfterMapping(String aSmallRNAsequenceAfterMapping) {
        smallRNAsequenceAfterMapping =
aSmallRNAsequenceAfterMapping.toUpperCase().replaceAll("T", "U");
    }

    /**
     * @return the precursorSequence
     */
    public  String getPrecursorSequence() {
        return precursorSequence;
    }

    /**
    * @return the precursorSequence lentgh
    */
    public  int getPrecursorSequenceLentgh() {
        return getPrecursorSequence().length();
    }

    /**
     * @param aPrecursorSequence the precursorSequence to set
     */
    public  void setPrecursorSequence(String aPrecursorSequence) {
        precursorSequence = aPrecursorSequence;
    }

    /**
     * @return the precursorStructure
     */
    public  String getPrecursorStructure() {
        return precursorStructure;
    }

    /**
     * @param aPrecursorStructure the precursorStructure to set
     */
    public  void setPrecursorStructure(String aPrecursorStructure) {
        precursorStructure = aPrecursorStructure;
    }

    /**
     * @return the precursorStructureMFE
     */
    public  double getPrecursorStructureMFE() {
        return precursorStructureMFE;
    }

    /**
     * @param aPrecursorStructureMFE the precursorStructureMFE to set
     */
    public  void setPrecursorStructureMFE(double aPrecursorStructureMFE) {
        precursorStructureMFE = aPrecursorStructureMFE;
    }

    /**
     * @return the predictorName
     */
    public  String getPredictorName() {
        return predictorName;
    }

    /**
     * @param aPredictorName the predictorName to set
     */
    public  void setPredictorName(String aPredictorName) {
        predictorName = aPredictorName;
    }

    /**
     * @return the predictorScore
     */
    public  String getPredictorScore() {
        return predictorScore;
    }
```

```
/**
 * @param aPredictorScore the predictorScore to set
 */
public  void setPredictorScore(String aPredictorScore) {
    predictorScore = aPredictorScore;
}

/**
 * @return the miRNA
 */
public  boolean isMiRNA() {
    return miRNA;
}

/**
 * @param aMiRNA the miRNA to set
 */
public  void setMiRNA(boolean aMiRNA) {
    miRNA = aMiRNA;
}

/**
 * @return the notes
 */
public  String getNotes() {
    return notes;
}

/**
 * @param aNotes the notes to set
 */
public  void setNotes(String aNotes) {
    notes = aNotes;
}

/**
 * @return the originalColorsequence
 */
public String getOriginalColorLibrary() {
    return originalColorLibrary;
}

/**
 * @param originalColorsequence the originalColorsequence to set
 */
public void setOriginalColorLibrary(String originalColorLibrary) {
    this.originalColorLibrary = originalColorLibrary;
}



/**
 * @return the ribosomal
 */
public boolean isRibosomal() {
    return ribosomal;
}

/**
 * @param ribosomal the ribosomal to set
 */
public void setRibosomal(boolean ribosomal) {
    this.ribosomal = ribosomal;
}

/**
 * @return the filter
 */
public String getFilter() {
    return filter;
}

/**
 * @param filter the filter to set
 */
public void setFilter(String filter) {
```

```
        this.filter = filter;
    }

    /**
     * @return the groupID
     */
    public String getGroupIDafterPrediction() {
        return groupIDafterPrediction;
    }

    /**
     * @param groupID the groupID to set
     */
    public void setGroupIDafterPrediction(String groupID) {
        this.groupIDafterPrediction = groupID;
    }

    /**
     * @return the groupName
     */
    public String getGroupNameAfterPrediction() {
        return groupNameafterPrediction;
    }

    /**
     * @param groupName the groupName to set
     */
    public void setGroupNameAfterPrediction(String groupName) {
        this.groupNameafterPrediction = groupName;
    }

    /**
     * @return the Familly
     */
    public String getFamily() {
        return Family;
    }

    /**
     * @param Familly the Familly to set
     */
    public void setFamily(String Familly) {
        this.Family = Familly;
    }

    public boolean haveExpression(){
        if (this.getExpressionPerLib()!=null){
            return true;
        }else return false;
    }



    /**
     * @return the ribosomalName
     */
    public String getRibosomalName() {
        return ribosomalName;
    }

    /**
     * @param ribosomalName the ribosomalName to set
     */
    public void setRibosomalName(String ribosomalName) {
        this.ribosomalName = ribosomalName;
    }

    /**
     * @return the targetgenesList
     */
    public String getTargetgenesListPtVirg() {
        return targetgenesList;
    }
```

```
    /**
     * @return the targetgenesList
     */
    public String getTargetgenesListTab() {
        if (getTargetgenesListPtVirg()!=null){
            return targetgenesList.replaceAll(";", "\t");
        }
        else return null;
    }

    /**
     * @param targetgenesList the targetgenesList to set
     */
    public void setTargetgenesList(String targetgenesList) {
        this.targetgenesList = targetgenesList;
    }


    public int getTargetGenesAmountFromList(){
        if (this.getTargetgenesListPtVirg()!=null){
            String targets[] = getTargetgenesListPtVirg().split(";");
            return targets.length;
        } else {
            return 0;
        }
    }


    /**
     * @return the predictorsCommon
     */
    public boolean isPredictorsCommon() {
        return predictorsCommon;
    }

    /**
     * @param predictorsCommon the predictorsCommon to set
     */
    public void setPredictorsCommon(boolean predictorsCommon) {
        this.predictorsCommon = predictorsCommon;
    }

    /**
     * @return the conserved
     */
    public boolean isConserved() {
        return conserved;
    }

    /**
     * @param conserved the conserved to set
     */
    public void setConserved(boolean conserved) {
        this.conserved = conserved;
    }

    /**
     * @return the smallRNAsequenceStarAfterMapping
     */
    public String getMirnaStar() {
        return mirnaStar;
    }

    /**
     * @param smallRNAsequenceStarAfterMapping the smallRNAsequenceStarAfterMapping to set
     */
    public void setMirnaStar(String smallRNAsequenceStarAfterMapping) {
        this.mirnaStar = smallRNAsequenceStarAfterMapping;
    }

    /**
     * @return the armOfSmallRNAOnPrecursor
     */
    public String getArmOfSmallRNAOnPrecursor() {
        return armOfSmallRNAOnPrecursor;
    }
```

```
/**
 * @param armOfSmallRNAOnPrecursor the armOfSmallRNAOnPrecursor to set
 */
public void setArmOfSmallRNAOnPrecursor(String armOfSmallRNAOnPrecursor) {
    this.armOfSmallRNAOnPrecursor = armOfSmallRNAOnPrecursor;
}

/**
 * @return the mirnaStarAbondance
 */
public int getMirnaStarAbondance() {
    return mirnaStarAbondance;
}

/**
 * @param mirnaStarAbondance the mirnaStarAbondance to set
 */
public void setMirnaStarAbondance(int mirnaStarAbondance) {
    this.mirnaStarAbondance = mirnaStarAbondance;
}

/**
 * @return the originalQualityRead
 */
public int getOriginalQualityRead() {
    return originalQualityRead;
}

/**
 * @param originalQualityRead the originalQualityRead to set
 */
public void setOriginalQualityRead(int originalQualityRead) {
    this.originalQualityRead = originalQualityRead;
}

/**
 * @return the numberOfDistinctPrecursors
 */
public int getNumberOfDistinctPrecursors() {
    return numberOfDistinctPrecursors;
}

/**
 * @param numberOfDistinctPrecursors the numberOfDistinctPrecursors to set
 */
public void setNumberOfDistinctPrecursors(int numberOfDistinctPrecursors) {
    this.numberOfDistinctPrecursors = numberOfDistinctPrecursors;
}

/**
 * @return the precursorStructureMFEcorrected
 */
public double getPrecursorStructureMFEcorrected() {
    return precursorStructureMFEcorrected;
}

/**
 * @param precursorStructureMFEcorrected the precursorStructureMFEcorrected to set
 */
public void setPrecursorStructureMFEcorrected(double precursorStructureMFEcorrected) {
    this.precursorStructureMFEcorrected = precursorStructureMFEcorrected;
}


/**
 * @return the expressionPerLibCorrected
 */
public int[] getExpressionPerLibCorrected() {
    return expressionPerLibCorrected;
}

/**
 * @param expressionPerLibCorrected the expressionPerLibCorrected to set
 */
public void setExpressionPerLibCorrected(int[] expressionPerLibCorrected) {
```

```
        this.expressionPerLibCorrected = expressionPerLibCorrected;
    }

    /**
     * @return the unirefsfromTargetGenes
     */
    public String getUnirefsfromTargetGenes() {
//        String tabs[]=this.getTargetgenesListPtVirg().split(";");
//        String unirefs="";
//        HashMap<String,Double> hm = new HashMap<String,Double>();
//        for (String tab : tabs) {
//            if(tab.contains("UniRef")){
//                int idx = tab.indexOf("UniRef")+10;
//                double scor=Double.valueOf(tab.split(",")[3]);
//                tab=tab.substring(idx);
//                String uniref=tab.substring(0, tab.indexOf(" "));
//                if (!hm.containsKey(uniref)){
//                    hm.put(uniref,scor);
//                }
//
//            }
//        }
//
//        for (String uni : hm.keySet()) {
//            unirefs=unirefs+uni+";";
//        }
//        if (unirefs.length()>0){
//            unirefsfromTargetGenes=unirefs.substring(0, unirefs.length()-1);
//        }

        return unirefsfromTargetGenes;
    }

    public LinkedHashMap sortHashMapByValuesD(HashMap passedMap) {
        List mapKeys = new ArrayList(passedMap.keySet());
        List mapValues = new ArrayList(passedMap.values());
        Collections.sort(mapValues);
        Collections.sort(mapKeys);

        LinkedHashMap sortedMap =
            new LinkedHashMap();

        Iterator valueIt = mapValues.iterator();
        while (valueIt.hasNext()) {
            Object val = valueIt.next();
            Iterator keyIt = mapKeys.iterator();

            while (keyIt.hasNext()) {
                Object key = keyIt.next();
                String comp1 = passedMap.get(key).toString();
                String comp2 = val.toString();

                if (comp1.equals(comp2)){
                    passedMap.remove(key);
                    mapKeys.remove(key);
                    sortedMap.put((String)key, (Double)val);
                    break;
                )

            )

        }
        return sortedMap;
    )

    /**
     * @param unirefsfromTargetGenes the unirefsfromTargetGenes to set
     */
    public void setUnirefsfromTargetGenes(String unirefsfromTargetGenes) {
        this.unirefsfromTargetGenes = unirefsfromTargetGenes;
    }

    /**
     * @return the GOProcessBiologic
```

```
     */
    public String getGOProcessBiologic() {
        return GOProcessBiologic;
    }

    /**
     * @param GOProcessBiologic the GOProcessBiologic to set
     */
    public void setGOProcessBiologic(String GOProcessBiologic) {
        this.GOProcessBiologic = GOProcessBiologic;
    }

    /**
     * @return the GOComponentCell
     */
    public String getGOComponentCell() {
        return GOComponentCell;
    }

    /**
     * @param GOComponentCell the GOComponentCell to set
     */
    public void setGOComponentCell(String GOComponentCell) {
        this.GOComponentCell = GOComponentCell;
    }

    /**
     * @return the GOFunctionMolecular
     */
    public String getGOFunctionMolecular() {
        return GOFunctionMolecular;
    }

    /**
     * @param GOFunctionMolecular the GOFunctionMolecular to set
     */
    public void setGOFunctionMolecular(String GOFunctionMolecular) {
        this.GOFunctionMolecular = GOFunctionMolecular;
    }

    /**
     * @return the GOpvalue
     */
    public String getGOproteinName() {
        return GOproteinName;
    }

    /**
     * @param GOpvalue the GOpvalue to set
     */
    public void setGOproteinName(String GOproteinName) {
        this.GOproteinName = GOproteinName;
    }

        /**
     * @return the GOproteinID
     */
    public String getGOproteinID() {
        return GOproteinID;
    }

    /**
     * @param GOproteinID the GOproteinID to set
     */
    public void setGOproteinID(String GOproteinID) {
        this.GOproteinID = GOproteinID;
    }

    /**
     * @return the nonCodant
     */
    public boolean isNonCodant() {
        return nonCodant;
    }

    /**
```

```
     * @param nonCodant the nonCodant to set
     */
    public void setNonCodant(boolean nonCodant) {
        this.nonCodant = nonCodant;
    }

        /**
     * @return the lowComplexity
     */
    public boolean isLowComplexity() {
        return lowComplexity;
    }

    /**
     * @param lowComplexity the lowComplexity to set
     */
    public void setLowComplexity(boolean lowComplexity) {
        this.lowComplexity = lowComplexity;
    }

    /**
     * @return the mappingMismatchsAdded
     */
    public boolean isMappingMismatchsAdded() {
        return mappingMismatchsAdded;
    }

    /**
     * @param mappingMismatchsAdded the mappingMismatchsAdded to set
     */
    public void setMappingMismatchsAdded(boolean mappingMismatchsAdded) {
        this.mappingMismatchsAdded = mappingMismatchsAdded;
    }



    /**
     * @return the mappingMismatchsCorrected
     */
    public int getMappingMismatchsCorrected() {
        return mappingMismatchsCorrected;
    }

    /**
     * @param mappingMismatchsCorrected the mappingMismatchsCorrected to set
     */
    public void setMappingMismatchsCorrected(int mappingMismatchsCorrected) {
        this.mappingMismatchsCorrected = mappingMismatchsCorrected;
    }

    /**
     * @return the mipredRealpseudo
     */
    public String getMipredRealpseudo() {
        return mipredRealpseudo;
    }

    /**
     * @param mipredRealpseudo the mipredRealpseudo to set
     */
    public void setMipredRealpseudo(String mipredRealpseudo) {
        this.mipredRealpseudo = mipredRealpseudo;
    }

    /**
     * @return the mipredConfidence
     */
    public double getMipredConfidence() {
        return mipredConfidence;
    }

    /**
     * @param mipredConfidence the mipredConfidence to set
     */
    public void setMipredConfidence(double mipredConfidence) {
        this.mipredConfidence = mipredConfidence;
```

```
}

public String toStringHeader(){
    String s=""+
            "ID"+
            "\tSmallRNA color Name"+
            "\tOriginal Color Library"+
            "\tRead Quality (QV<=10)"+
            "\tOriginal smallRNA sequence"+
            "\tExpression per lib"+
            "\tTotal Expression"+
            "\tExpression per lib corrected"+
            "\tTotal Expression corrected"+
            "\tSmallRNA sequence after mapping length"+
            "\tMapping mismatchs corrected"+
            "\tMapping mismatch added on missing nt"+
            "\tStart position on source"+
            "\tStrand on mapping source"+
            "\tSmallRNA sequence after mapping (miRNA)"+
            "\tNumber of distinct precursors"+
            "\tArm of SmallRNA on Precursor"+
            "\tmiRNA*"+
            //"\t"+this.getMirnaStarAbondance()+
            "\tMapping source accession"+
            //"\tMapping source name"+
            "\tPrecursor sequence"+
            "\tPrecursor structure"+
            "\tPrecursor structure MFE"+
            //"Precursor structure MFE"+this.getPrecursorStructureMFE()+
            "\tPrecursor sequence lentgh"+
            "\tMAQ mapper"+
            "\tPredictor Name"+
            "\tPredictor score"+
            "\tmiPred real/pseudo"+
            "\tmiPred confidence"+
            "\tCommon to both predictors"+
            //"\tIs a miRNA"+
            "\tCategory"+
            //"\t"+this.getNotes()+
            //"\t"+this.getFilter()+
            "\tConserved in mirnaDBs"+
            "\tRibosomal"+
            //"\t"+this.getRibosomalName()+
            "\tnonCodant"+
            "\tLow complexity"+
            "\tGroup ID"+
            //"\t"+this.getGroupNameAfterPrediction()+
            "\tFamily"+
            "\tTarget genes amount"+
            "\tGO Biological Processes"+
            "\tGO Molecular Functions"+
            "\tGO Component Cells"+
            "\tGO Protein IDs"+
            "\tGO Protein Names"+
            "\tGO Unirefs from target genes"+
            "\tTarget genes"
            ;
    return s;
}
    @Override
public String toString(){
    String s="";
    s=""+
            this.getId()+
            "\t"+this.getSmallRNAcolorName()+
            "\t"+this.getOriginalColorLibrary()+
            "\t"+this.getOriginalQualityRead()+
            "\t"+this.getOriginalSmallRNAsequence()+
            "\t"+this.getExpressionPerLibToStringVirgules()+
            "\t"+this.getExpressionTotal()+
            "\t"+this.getExpressionPerLibCorrectedToStringVirgules()+
            "\t"+this.getExpressionTotalCorrected()+
            "\t"+this.getSmallRNAsequenceAfterMappingWithNT().length()+
            "\t"+this.getMappingMismatchsCorrected()+
            "\t"+this.isMappingMismatchsAdded()+
```

```
                "\t"+this.getStartPositionOnSource()+
                "\t"+this.getStrandOnSource()+
                "\t"+this.getSmallRNAsequenceAfterMappingWithNTdone()+
                "\t"+this.getNumberOfDistinctPrecursors()+
                "\t"+this.getArmOfSmallRNAOnPrecursor()+
                "\t"+this.getMirnaStar()+
                //"\t"+this.getMirnaStarAbondance()+
                "\t"+this.getMappingSourceAccession()+
                //"\t"+this.getMappingSourceName()+
                "\t"+this.getPrecursorSequence()+
                "\t"+this.getPrecursorStructure()+
                //"\t"+this.getPrecursorStructureMFE()+
                "\t"+this.getPrecursorStructureMFEcorrected()+
                "\t"+this.getPrecursorSequenceLentgh()+
                "\t"+this.getMapper()+
                "\t"+this.getPredictorName()+
                "\t"+this.getPredictorScore()+
                "\t"+this.getMipredRealpseudo()+
                "\t"+this.getMipredConfidence()+
                "\t"+this.isPredictorsCommon()+
                //"\t"+this.isMiRNA()+
                "\t"+this.getCategory()+
                //"\t"+this.getNotes()+
                //"\t"+this.getFilter()+
                "\t"+this.isConserved()+
                "\t"+this.isRibosomal()+
                //"\t"+this.getRibosomalName()+
                "\t"+this.isNonCodant()+
                "\t"+this.isLowComplexity()+
                "\t"+this.getGroupIDafterPrediction()+
                //"\t"+this.getGroupNameAfterPrediction()+
                "\t"+this.getFamily()+
                "\t"+this.getTargetGenesAmountFromList()+
                "\t"+this.getGOProcessBiologic()+
                "\t"+this.getGOFunctionMolecular()+
                "\t"+this.getGOComponentCell()+
                "\t"+this.getGOproteinID()+
                "\t"+this.getGOproteinName()+
                "\t"+this.getUnirefsfromTargetGenes()+
                "\t"+this.getTargetgenesListPtVirg()
                ;
    return s.replaceAll("null", "NA");
}

public String toStringCommonsHeader(){
    String s=""+
                //"\tID"+
                //"SmallRNA color name"+
                //"\tOriginal Color Library"+
                "Read quality (QV<=10)"+
                "\tOriginal smallRNA sequence"+
                "\tExpression per lib"+
                "\tTotal Expression"+
                "\tExpression per lib corrected"+
                "\tTotal Expression corrected"+
                "\tSmallRNA sequence after mapping length"+
                "\tMapping mismatchs corrected"+
                "\tMapping mismatch added on missing nt"+
                "\tStart position on source"+
                "\tStrand on mapping source"+
                "\tSmallRNA sequence after mapping (miRNA)"+
                "\tNumber of distinct precursors"+
                "\tArm of SmallRNA on Precursor"+
                "\tmiRNA*"+
                //"\t"+this.getMirnaStarAbondance()+
                "\tMapping source accession"+
                //"\t"+this.getMappingSourceName()+
                "\tPrecursor sequence"+
                "\tPrecursor structure"+
                "\tPrecursor structure MFE"+
                //"Precursor structure MFE"+this.getPrecursorStructureMFE()+
                "\tPrecursor sequence lentgh"+
                "\tMAQ mapper"+
                "\tPredictor Name"+
                "\tPredictor score"+
                "\tmiPred real/pseudo"+
```

```
            "\tmiPred confidence"+
            //"\tCommon to both predictors"+
            "\tCategory"+
            //"\t"+this.getNotes()+
            //"\t"+this.getFilter()+
            "\tConserved in mirnaDBs"+
            //"\tRibosomal"+
            //"\t"+this.getRibosomalName()+
            //"\tnonCodant"+
            //"\tLow complexity"+
            //"\tGroup ID"+
            //"\t"+this.getGroupNameAfterPrediction()+
            "\tFamily"+
            "\tTarget genes amount"+
            "\tGO Biological Processes"+
            "\tGO Molecular Functions"+
            "\tGO Component Cells"+
            "\tGO Protein IDs"+
            "\tGO Protein Names"+
            "\tGO Unirefs from target genes"+
            "\tTarget genes"
            ;
    return s;
}

public String toStringCommons(){
    String s="";
    s=""+
            //this.getId()+
            //this.getSmallRNAcolorName()+
            //"\t"+this.getOriginalColorLibrary()+
            +this.getOriginalQualityRead()+
            "\t"+this.getOriginalSmallRNAsequence()+
            "\t"+this.getExpressionPerLibToStringVirgules()+
            "\t"+this.getExpressionTotal()+
            "\t"+this.getExpressionPerLibCorrectedToStringVirgules()+
            "\t"+this.getExpressionTotalCorrected()+
            "\t"+this.getSmallRNAsequenceAfterMappingWithNT().length()+
            "\t"+this.getMappingMismatchsCorrected()+
            "\t"+this.isMappingMismatchsAdded()+
            "\t"+this.getStartPositionOnSource()+
            "\t"+this.getStrandOnSource()+
            "\t"+this.getSmallRNAsequenceAfterMappingWithNTdone()+
            "\t"+this.getNumberOfDistinctPrecursors()+
            "\t"+this.getArmOfSmallRNAOnPrecursor()+
            "\t"+this.getMirnaStar()+
            //"\t"+this.getMirnaStarAbondance()+
            "\t"+this.getMappingSourceAccession()+
            //"\t"+this.getMappingSourceName()+
            "\t"+this.getPrecursorSequence()+
            "\t"+this.getPrecursorStructure()+
            //"\t"+this.getPrecursorStructureMFE()+
            "\t"+this.getPrecursorStructureMFEcorrected()+
            "\t"+this.getPrecursorSequenceLentgh()+
            "\t"+this.getMapper()+
            "\t"+this.getPredictorName()+
            "\t"+this.getPredictorScore().toString().replace(".", ",")+
            "\t"+this.getMipredRealpseudo()+
            "\t"+this.getMipredConfidence()+
            //"\t"+this.isPredictorsCommon()+
            "\t"+this.getCategory()+
            //"\t"+this.getNotes()+
            //"\t"+this.getFilter()+
            "\t"+this.isConserved()+
            //"\t"+this.isRibosomal()+
            //"\t"+this.getRibosomalName()+
            //"\t"+this.isNonCodant()+
            //"\t"+this.isLowComplexity()+
            //"\t"+this.getGroupIDafterPrediction()+
            //"\t"+this.getGroupNameAfterPrediction()+
            "\t"+this.getFamily()+
            "\t"+this.getTargetGenesAmountFromList()+
            "\t"+this.getGOProcessBiologic()+
            "\t"+this.getGOFunctionMolecular()+
            "\t"+this.getGOComponentCell()+
            "\t"+this.getGOproteinID()+
```

```
                "\t"+this.getGOproteinName()+
                "\t"+this.getUnirefsfromTargetGenes()+
                "\t"+this.getTargetgenesListPtVirg()
                ;
        return s.replaceAll("null", "NA");
    }


    public String toStringRNAfoldfasta(){
        String s="";
        s=">"+
                this.getId()+
                "\t"+this.getSmallRNAsequenceAfterMappingWithNTdone()+
                "\n"+this.getPrecursorSequence();
        return s;
    }

    public String toStringSmallRNAfasta(){
        String s="";
        s=">"+
                this.getId()+
                "\n"+this.getSmallRNAsequenceAfterMappingWithNTdone();
        return s;
    }

    public String toStringFolding(){
        String s="";
        s=""+
                this.getSmallRNAsequenceAfterMappingWithNTdone()+
                "\t"+this.getId()+
                "\t"+this.getPrecursorSequence();
        return s;
    }

    public String toStringExpressionProfiles(){
        return
this.getSmallRNAsequenceAfterMappingWithNTdone()+"\t"+this.getExpressionPerLibToStringTabs()+"\t"
+this.getExpressionTotal();
    }

    /**
     * @return the category
     */
    public char.getCategory() {
        return category;
    }

    /**
     * @param category the category to set
     */
    public void setCategory(char category) {
        this.category = category;
    }


}
```

tools.java

Tools regroupe quelques outils souvent fait appel dans les scripts. Il regroupe plusieurs fonction telles que :

- **Compter le nombre de contigs/éléments d'un fichier FASTA**
- **Compter le nombre de ligne très rapidement**
- **Calculer le complément de l'ADN**
- **Calculer le reverse complément de l'ADN**
- **Affiche la mémoire du système**
- **Lancer une commande système sous linux (bash)**
- **Lancer une commande système sous windows (shell)**
- **Lancer une commande système sous cygwin**
- **Permet de passer d'un code couleur à une séquence nucléotidique**
- **Permet de passer d'une séquence nucléotidique à un code couleur**

```
/*
 * Tools for version2mirnas
```

```
 */

package tools;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.RandomAccessFile;
import java.util.HashMap;

/**
 *
 * @author Mickael
 */
public class tools {

    public static HashMap<String,Integer> hm = new HashMap<String, Integer>();
    public static HashMap<String,Character> hmc = new HashMap<String, Character>();
    /**
     * Calcule le nombre de contigs présents et l'affiche
     * @param genome
     * @return nombre de contigs
     */
    public static int check_number_contigs(File genome) {
        //System.out.println("Analysing genome size of "+genome+"...");
        int count=0;
        try {
            BufferedReader br=new BufferedReader(new FileReader(genome));
            while (br.ready()) {
                if (br.readLine().startsWith(">")){
                    count++;
                }
            }
            br.close();
         } catch (Exception e) {
            e.printStackTrace();
            return 500000;
        }
        //System.out.println("Total sequences : "+count);
        return count;
    }

    /**
     * Complement a string of IUPAC DNA nucleotides (output A C T G only).
     *
     * @param s The string of one-letter upper or lower case IUPAC nucleotides to complement.
     * @return A complemented string, ie A->T, T->A, C->G, G->C, U->A. Case is preserved.
     */
    public static String dnaComplement( String s )
    {
            char cgene[]= s.toCharArray();

            //complement
            int i;
            for( i=0; i < cgene.length; i ++ )
            {
                    switch( cgene[ i ])
                    {
                            case 'A': cgene[ i ]= 'T'; break;
                            case 'T': cgene[ i ]= 'A'; break;
                            case 'U': cgene[ i ]= 'A'; break;
                            case 'C': cgene[ i ]= 'G'; break;
                            case 'G': cgene[ i ]= 'C'; break;
                            case 'a': cgene[ i ]= 't'; break;
                            case 't': cgene[ i ]= 'a'; break;
                            case 'u': cgene[ i ]= 'a'; break;
                            case 'c': cgene[ i ]= 'g'; break;
                            case 'g': cgene[ i ]= 'c'; break;
                    }

            }
```

```
            return new String( cgene );
    }

    /**
     * Reverse and complement a string of IUPAC DNA nucleotides (A C T G only).
     *
     * @param s The string of one-letter upper or lower case IUPAC nucleotides to reverse and
complement.
     * @return The reverse-complemented string, ie A->T, T->A, C->G, G->C. Case is preserved.
     */
    public static String dnaReverseComplement( String s )
    {
            StringBuffer r= new StringBuffer( dnaComplement( s ) );
            return r.reverse().toString();
    }


    static public boolean deleteDirectory(File path) {
        if( path.exists() ) {
          File[] files = path.listFiles();
          for(int i=0; i<files.length; i++) {
             if(files[i].isDirectory()) {
               deleteDirectory(files[i]);
             }
             else {
               files[i].delete();
             }
          }
        }
       return( path.delete() );
  }

    public static int countLines(File filename) {
        int count = 0;
        try {
            InputStream is = new BufferedInputStream(new FileInputStream(filename));
            byte[] c = new byte[1024];

            int readChars = 0;
            while ((readChars = is.read(c)) != -1) {
                for (int i = 0; i < readChars; ++i) {
                    if (c[i] == '\n') {
                        ++count;

                    }
                }
            }
        } catch (IOException iOException) {
        }
        return count;
    }

    public static String getColorLineInLibraries (int lineNumber, int mirnaNumber, File f){
        String color = null;
        try {
            FileInputStream fs= new FileInputStream(f);
            BufferedReader br = new BufferedReader(new InputStreamReader(fs));

            for(int i = 0; i < lineNumber; ++i) br.readLine();
            int mirna = Integer.valueOf(br.readLine().split("_")[1]);
            if (mirna!=mirnaNumber) System.err.println("mirna number not match
"+mirna+"!="+mirnaNumber);
            color = br.readLine();
            br.close();
        } catch (Exception e) {
        }
        return color;
    }

        /**
     * Print to System.out current Memory Allocation and Total System Core
     */
    public static String PrintMemory() {
        int Mb=1048576;
```

```
        String stri="System allocated memory: "+Runtime.getRuntime().totalMemory()/Mb+" MB System
free memory: "+Runtime.getRuntime().freeMemory()/Mb+" MB\n"+
                    "System total core: "+Runtime.getRuntime().availableProcessors()+"\n";

         return stri;
    }

   public static void CleanMemory() {
         Runtime r = Runtime.getRuntime();
         r.gc();
   }

       /**
     * Execute a command in linux with bash
     * @param infile
     * @param outfile
     */
        public static void executeLinuxCommand (String cmd){

        try {
            //System.out.println(cmd+"\n");
            ProcessBuilder pb = new ProcessBuilder("bash", "-c", cmd);
            pb.redirectErrorStream(true); // use this to capture messages sent to stderr
            Process shell = pb.start();
            InputStream shellIn = shell.getInputStream(); // this captures the output from the
command
            int shellExitStatus = shell.waitFor(); // wait for the shell to finish and get the
return code
            // at this point you can process the output issued by the command
            // for instance, this reads the output and writes it to System.out:
            int c;
            while ((c = shellIn.read()) != -1) {
                System.out.write(c);
            }

            // close the stream

            shellIn.close();
        } catch (Exception e) {

        }
    }


        /**
         * Execute windows command with cmd
         * @param cmd
         */
    public static void executeWindowsCommand(String cmd) {

        System.out.println("Removing duplicates...");

        System.out.println(cmd+"\n");
        Runtime runtime = Runtime.getRuntime();

        try{
            //Process ps=pb.start();
            Process ps=runtime.exec(cmd);
            InputStream is=ps.getInputStream();

            BufferedReader bri=new BufferedReader(new InputStreamReader(is));
            String line;
            while ((line=bri.readLine())!=null) {
               System.out.println(line);
            }
//           ps.waitFor();
        }
        catch(Exception e){
            e.printStackTrace();
        }
        System.out.println("End");
    }

            /**
         * Execute cygwin command with cmd
         * @param cmd
```

```
     */
    public static void executeCygwinCommand(String cmd) {

        //System.out.println("Removing duplicates...");

        System.out.println(cmd+"\n");

        try{
            Process p = Runtime.getRuntime().exec(cmd, new String[] { "PATH=C:\\cygwin\\bin" });
            //Process ps=pb.start();
            InputStream is=p.getInputStream();

            BufferedReader bri=new BufferedReader(new InputStreamReader(is));
            String line;
            while ((line=bri.readLine())!=null) {
               System.out.println(line);
            }
//           ps.waitFor();
        }
        catch(Exception e){
            e.printStackTrace();
        }
        System.out.println("End");
    }

    public static void createColorHashmap() {
        hm.put("AA", 0);
        hm.put("AC", 1);
        hm.put("AG", 2);
        hm.put("AT", 3);
        hm.put("AN", 4);

        hm.put("CA", 1);
        hm.put("CC", 0);
        hm.put("CG", 3);
        hm.put("CT", 2);
        hm.put("CN", 4);

        hm.put("GA", 2);
        hm.put("GC", 3);
        hm.put("GG", 0);
        hm.put("GT", 1);
        hm.put("GN", 4);

        hm.put("TA", 3);
        hm.put("TC", 2);
        hm.put("TG", 1);
        hm.put("TT", 0);
        hm.put("TN", 4);

        hm.put("NA", 4);
        hm.put("NC", 4);
        hm.put("NG", 4);
        hm.put("NT", 4);
        hm.put("NN", 0);
    }

    public static void createTranslateColorHashmap() {
        hmc.put("A0", 'A');
        hmc.put("A1", 'C');
        hmc.put("A2", 'G');
        hmc.put("A3", 'T');

        hmc.put("C0", 'C');
        hmc.put("C1", 'A');
        hmc.put("C2", 'T');
        hmc.put("C3", 'G');

        hmc.put("G0", 'G');
        hmc.put("G1", 'T');
        hmc.put("G2", 'A');
        hmc.put("G3", 'C');

        hmc.put("T0", 'T');
        hmc.put("T1", 'G');
        hmc.put("T2", 'C');
```

```
        hmc.put("T3", 'A');
    }
}
```

BIBLIOGRAPHIE

Aalst, W. v. d. & Hee, K. M. v. (2004), *Workflow management : models, methods, and systems*, MIT Press, Cambridge, Mass.

Afgan, E., Baker, D., Coraor, N., Chapman, B., Nekrutenko, A. & Taylor, J. (2010), 'Galaxy cloudman: delivering cloud compute clusters', *BMC bioinformatics* **11 Suppl 12**, S4.

Alberts, B. (2002), *Molecular biology of the cell*, 4th edn, Garland Science, New York.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. (1990), 'Basic local alignment search tool', *J Mol Biol* **215**(3), 403–10.

Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W. & Lipman, D. J. (1997), 'Gapped blast and psi-blast: a new generation of protein database search programs', *Nucleic Acids Res* **25**(17), 3389–402.

Ambros, V. (1989), 'A hierarchy of regulatory genes controls a larva-to-adult developmental switch in c. elegans', *Cell* **57**(1), 49–57.

Babak, T., Zhang, W., Morris, Q., Blencowe, B. J. & Hughes, T. R. (2004), 'Probing microrna with microarrays: tissue specificity and functional inference', *Rna* **10**(11), 1813–9.

Bairoch, A., Apweiler, R., Wu, C. H., Barker, W. C., Boeckmann, B., Ferro, S., Gasteiger, E., Huang, H., Lopez, R., Magrane, M., Martin, M. J., Natale, D. A., O'Donovan, C., Redaschi, N. & Yeh, L. S. (2005), 'The universal protein resource (uniprot)', *Nucleic Acids Res* **33**(Database issue), D154–9.

Beaulah, S. A., Correll, M. A., Munro, R. E. & Sheldon, J. G. (2008), 'Addressing informatics challenges in translational research with workflow technology', *Drug Discov Today* **13**(17-18), 771–7.

Benjamini, Y. & Hochberg, Y. (1995), 'Controlling the false discovery rate - a practical and powerful approach to multiple testing', *Journal of the Royal Statistical Society Series B-Methodological* **57**(1), 289–300.

Berne, C., Pignol, D., Lavergne, J. & Garcia, D. (2007), 'Cyp201a2, a cytochrome p 450 from rhodopseudomonas palustris, plays a key role in the biodegradation of tributyl phosphate', *Applied microbiology and biotechnology* **77**(1), 135–144.

Betel, D., Wilson, M., Gabow, A., Marks, D. S. & Sander, C. (2008), 'The microrna.org resource: targets and expression', *Nucleic Acids Res* **36**(Database issue), D149–53.

Birnstiel, M. & Schaufele, F. (1988), 'Structure and function of minor snrnps', *Structure and Function of Major and Minor Small Nuclear Ribonucleoprotein Particles. ML Birnstiel, editor. Springer-Verlag New York Inc., New York* **155**, 182.

Bonnet, E., He, Y., Billiau, K. & Van de Peer, Y. (2010), 'Tapir, a web server for the prediction of plant microrna targets, including target mimics', *Bioinformatics* **26**(12), 1566–8.

Boualem, A., Laporte, P., Jovanovic, M., Laffont, C., Plet, J., Combier, J. P., Niebel, A., Crespi, M. & Frugier, F. (2008), 'Micrornal66 controls root and nodule development in medicago truncatula', *Plant J* **54**(5), 876–87.

Bowers, S., McPhillips, T., Riddle, S., Anand, M. K. & Ludascher, B. (2008), 'Kepler/ppod: Scientific workflow and provenance support for assembling the tree of life', *Provenance and Annotation of Data and Processes* **5272**, 70–77.

Breiman, L. (2001), 'Random forests', *Machine learning* **45**(1), 5–32.

Burge, C. & Karlin, S. (1997), 'Prediction of complete gene structures in human genomic dna1', *Journal of molecular biology* **268**(1), 78–94.

Burrows, M. & Wheeler, D. J. (1994), *A block-sorting lossless data compression algorithm*, Digital, Systems Research Center, Palo Alto, Calif.

Califano, A. & Rigoutsos, I. (1993), 'Flash: a fast look-up algorithm for string homology', *Proceedings / ... International Conference on Intelligent Systems for Molecular Biology ; ISMB. International Conference on Intelligent Systems for Molecular Biology* **1**, 56–64.

Cao, X., Yeo, G., Muotri, A. R., Kuwabara, T. & Gage, F. H. (2006), 'Noncoding rnas in the mammalian central nervous system', *Annu Rev Neurosci* **29**, 77–103.

Castanotto, D. & Rossi, J. J. (2009), 'The promises and pitfalls of rna-interference-based therapeutics', *Nature* **457**(7228), 426–33.

Cavalier-Smith, T. (2004), 'Only six kingdoms of life', *Proc Biol Sci* **271**(1545), 1251–62.

Chalfie, M., Horvitz, H. & Sulston, J. (1981), 'Mutations that lead to reiterations in the cell lineages of c. elegans', *Cell* **24**(1), 59–69.

Chang, C. & Lin, C. (2011), 'Libsvm: a library for support vector machines', *ACM Transactions on Intelligent Systems and Technology (TIST)* **2**(3), 27.

Chen, H.M., Li, Y.H., Wu, S.H. (2007), 'Bioinformatic prediction and experimental validation of a microRNA-directed tandem trans-acting siRNA cascade in Arabidopsis', *Proceedings of the National Academy of Sciences of the United States of America* **104**, 3318-23.

Chen, J. F., Mandel, E. M., Thomson, J. M., Wu, Q., Callis, T. E., Hammond, S. M., Conlon, F. L. & Wang, D. Z. (2006), 'The role of microrna-1 and microrna-133 in skeletal muscle proliferation and differentiation', *Nat Genet* **38**(2), 228–33.

Chen, X. M. (2004), 'A microrna as a translational repressor of apetala2 in arabidopsis flower development', *Science* **303**(5666), 2022–2025.

Cloonan, N., Forrest, A. R., Kolle, G., Gardiner, B. B., Faulkner, G. J., Brown, M. K., Taylor, D. F., Steptoe, A. L., Wani, S., Bethel, G., Robertson, A. J., Perkins, A. C., Bruce, S. J., Lee, C. C., Ranade, S. S., Peckham, H. E., Manning, J. M., McKernan, K. J. & Grimmond, S. M. (2008), 'Stem cell transcriptome profiling via massive-scale mrna sequencing', *Nat Methods* **5**(7), 613–9.

Clop, A., Marcq, F., Takeda, H., Pirottin, D., Tordoir, X., Bibe, B., Bouix, J., Caiment, F., Elsen, J. M., Eychenne, F., Larzul, C., Laville, E., Meish, F., Milenkovic, D., Tobin, J., Charlier, C. & Georges, M. (2006), 'A mutation creating a potential illegitimate microrna target site in the myostatin gene affects muscularity in sheep', *Nat Genet* **38**(7), 813–8.

Cohen, S. M. & Brennecke, J. (2006), 'Developmental biology. mixed messages in early development', *Science* **312**(5770), 65–6.

Cooper, T. A., Wan, L. & Dreyfuss, G. (2009), 'Rna and disease', *Cell* **136**(4), 777–793.

Dai, X. & Zhao, P. X. (2011), 'psrnatarget: a plant small rna target analysis server', *Nucleic Acids Res* **39**(Web Server issue), W155–9.

Dardel, F. & Kapas, F. (2002), *Bioinformatique : genomique et post-genomique*, Editions de l'ecole polytechnique, Palaiseau [France].

Darwin, C. R. (1859), *On the origin of species*, Cambr., Mass.

Delhaize, E., Taylor, P., Hocking, P. J., Simpson, R. J., Ryan, P. R. & Richardson, A. E. (2009), 'Transgenic barley (hordeum vulgare l.) expressing the wheat aluminium resistance gene (taalmt1) shows enhanced phosphorus nutrition and grain production when grown on an acid soil', *Plant biotechnology journal* **7**(5), 391–400.

Dinov, I. D., Torri, F., Macciardi, F., Petrosyan, P., Liu, Z., Zamanyan, A., Eggert, P., Pierce, J., Genco, A., Knowles, J. A., Clark, A. P., Van Horn, J. D., Ames, J., Kesselman, C. & Toga, A. W. (2011), 'Applications of the pipeline environment for visual informatics and genomics computations', *BMC bioinformatics* **12**(1), 304.

Dirks, R. M., Lin, M., Winfree, E. & Pierce, N. A. (2004), 'Paradigms for computational nucleic acid design', *Nucleic Acids Res* **32**(4), 1392–403.

Eddy, S. R. & Durbin, R. (1994), 'Rna sequence-analysis using covariance-models', *Nucleic Acids Research* **22**(11), 2079–2088.

Elles, R. & Mountford, R. (2004), *Molecular diagnosis of genetic diseases*, Humana Press, Totowa, N.J.

Farrar, M. (2007), 'Striped smith-waterman speeds database searches six times over other simd implementations', *Bioinformatics* **23**(2), 156–61.

Fontana, P., DemattÃ¨, L., Cestaro, A., Segala, C., Velasco, R. & Toppo, S. (2006), 'Goretriever: a novel gene ontology annotation tool based on semantic similarity for knowledge discovery in database', *Proceedings of Bioinformatics Italian Society* .

Franco-Zorrilla, J. M., Valli, A., Todesco, M., Mateos, I., Puga, M. I., Rubio-Somoza, I., Leyva, A., Weigel, D., Garcia, J. A. & Paz-Ares, J. (2007), 'Target mimicry provides a new mechanism for regulation of microrna activity', *Nat Genet* **39**(8), 1033–7.

Friedlander, M. R., Chen, W., Adamidi, C., Maaskola, J., Einspanier, R., Knespel, S. & Rajewsky, N. (2008), 'Discovering micrornas from deep sequencing data using mirdeep', *Nature biotechnology* **26**(4), 407–415.

Friedlander, M. R., Mackowiak, S. D., Li, N., Chen, W. & Rajewsky, N. (2012), 'mirdeep2 accurately identifies known and hundreds of novel microrna genes in seven animal clades', *Nucleic Acids Res* **40**(1), 37–52.

Fujii, H., Chiou, T. J., Lin, S. I., Aung, K. & Zhu, J. K. (2005), 'A mirna involved in phosphate-starvation response in arabidopsis', *Curr Biol* **15**(22), 2038–43.

Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P., Zhang, Y., Blankenberg, D., Albert, I., Taylor, J., Miller, W., Kent, W. J. & Nekrutenko, A. (2005), 'Galaxy: a platform for interactive large-scale genome analysis', *Genome Res* **15**(10), 1451–5.

Gilbert, W. (1986), 'Origin of life: The rna world', *Nature* **319**(6055).

Goderis, A. & University of Manchester. School of computer, s. (2008), *Workflow re-use and discovery in bioinformatics*, University of Manchester, Manchester.

Gonzalez, F. J. & Nebert, D. W. (1990), 'Evolution of the p450-gene superfamily - animal plant warfare, molecular drive and human genetic-differences in drug oxidation', *Trends in Genetics* **6**(6), 182–186.

Grey, F., Antoniewicz, A., Allen, E., Saugstad, J., McShea, A., Carrington, J. C. & Nelson, J. (2005), 'Identification and characterization of human cytomegalovirus-encoded micrornas', *J Virol* **79**(18), 12095–9.

Griffiths-Jones, S. (2004), 'The microrna registry', *Nucleic Acids Res* **32**(Database issue), D109–11.

Griffiths-Jones, S., Grocock, R. J., van Dongen, S., Bateman, A. & Enright, A. J. (2006), 'mirbase: microrna sequences, targets and gene nomenclature', *Nucleic Acids Res* **34**(Database issue), D140–4.

Griffiths-Jones, S., Moxon, S., Marshall, M., Khanna, A., Eddy, S. R. & Bateman, A. (2005), 'Rfam: annotating non-coding rnas in complete genomes', *Nucleic Acids Res* **33**(Database issue), D121–4.

Griffiths-Jones, S., Saini, H. K., van Dongen, S. & Enright, A. J. (2008), 'mirbase: tools for microrna genomics', *Nucleic Acids Res* **36**(Database issue), D154–8.

Guo, H. S., Xie, Q., Fei, J. F. & Chua, N. H. (2005), 'Microrna directs mrna cleavage of the transcription factor nac1 to downregulate auxin signals for arabidopsis lateral root development', *Plant Cell* **17**(5), 1376–86.

Guy, C. L. (1990), 'Cold accelimation and freezing stress tolerance: Role of protein metabolism', *Annual Review of Plant Biology* **41**(1), 187–223.

Hackenberg, M., Rodriguez-Ezpeleta, N. & Aransay, A. M. (2011), 'miranalyzer: an update on the detection and analysis of micrornas in high-throughput sequencing experiments', *Nucleic Acids Res* **39**(Web Server issue), W132–8.

Hackenberg, M., Sturm, M., Langenberger, D., Falcon-Perez, J. M. & Aransay, A. M. (2009), 'miranalyzer: a microrna detection and analysis tool for next-generation sequencing experiments', *Nucleic Acids Res* **37**(Web Server issue), W68–76.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I. (2009), 'The weka data mining software: an update', *ACM SIGKDD Explorations Newsletter* **11**(1), 10–18.

Hamilton, A. J. & Baulcombe, D. C. (1999), 'A species of small antisense rna in posttranscriptional gene silencing in plants', *Science* **286**(5441), 950–2.

Hammell, M. (2010), 'Computational methods to identify mirna targets', *Semin Cell Dev Biol* **21**(7), 738–44.

Hipp, D. (2003), 'Sqlite', *www.sqlite.org* .

Hofacker, I. (2004), 'Rna secondary structure analysis using the vienna rna package', *Current Protocols in Bioinformatics* .

Hofacker, I. L., Fontana, W., Stadler, P. F., Bonhoeffer, L. S., Tacker, M. & Schuster, P. (1994), 'Fast folding and comparison of rna secondary structures', *Monatshefte Fur Chemie* **125**(2), 167–188.

Hofacker, I. L. & Stadler, P. F. (2006), 'Memory efficient folding algorithms for circular rna secondary structures', *Bioinformatics* **22**(10), 1172–6.

Homer, N., Merriman, B. & Nelson, S. F. (2009), 'Bfast: an alignment tool for large scale genome resequencing', *PloS one* **4**(11), e7767.

Hsieh, L. C., Lin, S. I., Shih, A. C., Chen, J. W., Lin, W. Y., Tseng, C. Y., Li, W. H. & Chiou, T. J. (2009), 'Uncovering small rna-mediated responses to phosphate deficiency in arabidopsis by deep sequencing', *Plant Physiol* **151**(4), 2120–32.

Huang, T. H., Fan, B., Rothschild, M. F., Hu, Z. L., Li, K. & Zhao, S. H. (2007), 'Mirfinder: an improved approach and software implementation for genome-wide fast microrna precursor scans', *BMC bioinformatics* **8**(1), 341.

Huang, X. & Madan, A. (1999), 'Cap3: A dna sequence assembly program', *Genome Res* **9**(9), 868–77.

Jacobsen, A., Krogh, A., Kauppinen, S. & Lindow, M. (2010), 'mirmaid: a unified programming interface for microrna data resources', *BMC bioinformatics* **11**(1), 29.

Jasinski, S., Vialette-Guiraud, A. C. & Scutt, C. P. (2010), 'The evolutionary-developmental analysis of plant micrornas', *Philos Trans R Soc Lond B Biol Sci* **365**(1539), 469–76.

Jiang, P., Wu, H., Wang, W., Ma, W., Sun, X. & Lu, Z. (2007), 'Mipred: classification of real and pseudo microrna precursors using random forest prediction model with combined features', *Nucleic Acids Research* **35**, W339–W344.

Jin, W., Grant, J. R., Stothard, P., Moore, S. S. & Guan, L. L. (2009), 'Characterization of bovine mirnas by sequencing and bioinformatics analysis', *BMC Mol Biol* **10**(1), 90.

Jin, W., Li, N., Zhang, B., Wu, F., Li, W., Guo, A. & Deng, Z. (2008), 'Identification and verification of microrna in wheat (triticum aestivum)', *J Plant Res* **121**(3), 351–5.

John, B., Enright, A. J., Aravin, A., Tuschl, T., Sander, C. & Marks, D. S. (2004), 'Human microrna targets', *PLoS Biol* **2**(11), e363.

Jones-Rhoades, M. W. (2010), 'Prediction of plant mirna genes', *Methods Mol Biol* **592**, 19–30.

Jou, W., Haegeman, G., Ysebaert, M. & Fiers, W. (1972), 'Nucleotide sequence of the gene coding for the bacteriophage ms2 coat protein', *Nature* **237**, 82–88.

Jurka, J. (1998), 'Repeats in genomic dna: mining and meaning', *Curr Opin Struct Biol* **8**(3), 333–7.

Kadri, S., Hinman, V. & Benos, P. V. (2009), 'Hhmmir: efficient de novo prediction of micrornas using hierarchical hidden markov models', *BMC bioinformatics* **10**(Suppl 1), S35.

Kal, A. J., van Zonneveld, A. J., Benes, V., van den Berg, M., Koerkamp, M. G., Albermann, K., Strack, N., Ruijter, J. M., Richter, A., Dujon, B., Ansorge, W. & Tabak, H. F. (1999), 'Dynamics of gene expression revealed by comparison of serial analysis of gene expression transcript profiles from yeast grown on two different carbon sources', *Mol Biol Cell* **10**(6), 1859–72.

Kim, J., Jung, J. H., Reyes, J. L., Kim, Y. S., Kim, S. Y., Chung, K. S., Kim, J. A., Lee, M., Lee, Y., Narry Kim, V., Chua, N. H. & Park, C. M. (2005), 'microrna-directed cleavage of athb15 mrna regulates vascular development in arabidopsis inflorescence stems', *Plant J* **42**(1), 84–94.

Kim, S. K., Nam, J. W., Rhee, J. K., Lee, W. J. & Zhang, B. T. (2006), 'mitarget: microrna target gene prediction using a support vector machine', *BMC bioinformatics* **7**(1), 411.

Kiss, T. (2001), 'Review: Small nucleolar rna-guided post-transcriptional modification of cellular rnas', *The EMBO Journal* **20**(14), 3617.

Krek, A., Grun, D., Poy, M. N., Wolf, R., Rosenberg, L., Epstein, E. J., MacMenamin, P., da Piedade, I., Gunsalus, K. C., Stoffel, M. & Rajewsky, N. (2005), 'Combinatorial microrna target predictions', *Nat Genet* **37**(5), 495–500.

Kurtz, S. (2003), 'The vmatch large scale sequence analysis software', *Ref Type: Computer Program* pp. 4–12.

Lagos-Quintana, M., Rauhut, R., Lendeckel, W. & Tuschl, T. (2001), 'Identification of novel genes coding for small expressed rnas', *Science* **294**(5543), 853–8.

Lai, E. C., Tomancak, P., Williams, R. W. & Rubin, G. M. (2003), 'Computational identification of drosophila microrna genes', *Genome Biol* **4**(7), R42.

Lamoril, J., Bouizegarane, P. & Bogard, M. (2010), 'Le monde complexe et mouvant des arn. seconde partieÂ : les microarns', *Immuno-analyse & Biologie Specialisee* **25**(5-6), 219–240.

Langmead, B., Trapnell, C., Pop, M. & Salzberg, S. L. (2009), 'Ultrafast and memory-efficient alignment of short dna sequences to the human genome', *Genome Biol* **10**(3), R25.

Larkin, M., Blackshields, G., Brown, N., Chenna, R., McGettigan, P., McWilliam, H., Valentin, F., Wallace, I., Wilm, A., Lopez, R. et al. (2007), 'Clustalw and clustalx version 2', *Bioinformatics* **23**(21), 2947–2948.

Lauter, N., Kampani, A., Carlson, S., Goebel, M. & Moose, S. P. (2005), 'microrna172 down-regulates glossy15 to promote vegetative phase change in maize', *Proc Natl Acad Sci U S A* **102**(26), 9412–7.

Lee, R. C., Feinbaum, R. L. & Ambros, V. (1993), 'The c. elegans heterochronic gene lin-4 encodes small rnas with antisense complementarity to lin-14', *Cell* **75**(5), 843–54.

Lehninger, L., Nelson, D. & Cox, M. (1993), *Principles of biochemistry, 2nd edition*, Worth.

Letunic, I. & Bork, P. (2007), 'Interactive tree of life (itol): an online tool for phylogenetic tree display and annotation', *Bioinformatics* **23**(1), 127–8.

Leung, W., Yiu, S., Cheung, D., Lai, L., Lin, M. & Kung, H. (2007), 'Computational prediction on mammalian and viral micrornas-a review', *International Journal of Integrative Biology* **1**(2), 118.

Lewis, B. P., Shih, I. H., Jones-Rhoades, M. W., Bartel, D. P. & Burge, C. B. (2003), 'Prediction of mammalian microrna targets', *Cell* **115**(7), 787–98.

Li, H., Ruan, J. & Durbin, R. (2008), 'Mapping short dna sequencing reads and calling variants using mapping quality scores', *Genome research* **18**(11), 1851–1858.

Li, J., Iwamoto, S., Sugimoto, N., Okuda, H. & Kajii, E. (1997), 'Dinucleotide repeat in the 3' flanking region provides a clue to the molecular evolution of the duffy gene', *Hum Genet* **99**(5), 573–7.

Li, R., Li, Y., Kristiansen, K. & Wang, J. (2008), 'Soap: short oligonucleotide alignment program', *Bioinformatics* **24**(5), 713–4.

Li, S. C., Shiau, C. K. & Lin, W. C. (2008), 'Vir-mir db: prediction of viral microrna candidate hairpins', *Nucleic Acids Res* **36**(Database issue), D184–9.

Lindow, M., Jacobsen, A., Nygaard, S., Mang, Y. & Krogh, A. (2007), 'Intragenomic matching reveals a huge potential for mirna-mediated regulation in plants', *PLoS Comput Biol* **3**(11), e238.

Lindsay, M. A. (2008), 'micrornas and the immune response', *Trends Immunol* **29**(7), 343–51.

Liu, C., Bai, B., SkogerbÃ„, G., Cai, L., Deng, W., Zhang, Y., Bu, D., Zhao, Y. & Chen, R. (2005), 'Noncode: an integrated knowledge database of non-coding rnas. www.noncode.org', *Nucleic acids research* **33**(suppl 1), D112.

Lord, E., Leclercq, M., Boc, A., Diallo, A. B. & Makarenkov, V. (2012), 'Armadillo 1.1: an original workflow platform for designing and conducting phylogenetic analysis and simulations', *PloS one* **7**(1), e29903.

Lu, C., Tej, S. S., Luo, S., Haudenschild, C. D., Meyers, B. C. & Green, P. J. (2005), 'Elucidation of the small rna component of the transcriptome', *Science* **309**(5740), 1567–9.

Major, F., Turcotte, M., Gautheret, D., Lapalme, G., Fillion, E. & Cedergren, R. (1991), 'The combination of symbolic and numerical computation for three-dimensional modeling of rna', *Science* **253**(5025), 1255–60.

Mallory, A. C. & Vaucheret, H. (2006), 'Functions of micrornas and related small rnas in plants', *Nat Genet* **38 Suppl**(1), S31–6.

Maragkakis, M., Reczko, M., Simossis, V. A., Alexiou, P., Papadopoulos, G. L., Dalamagas, T., Giannopoulos, G., Goumas, G., Koukis, E., Kourtis, K., Vergoulis, T., Koziris, N., Sellis, T., Tsanakas, P. & Hatzigeorgiou, A. G. (2009), 'Diana-microt web server: elucidating microrna functions through target prediction', *Nucleic Acids Research* **37**(suppl 2), W273–W276.

Mardis, E. R. (2008), 'The impact of next-generation sequencing technology on genetics', *Trends Genet* **24**(3), 133–41.

Mathews, C. K. & Van Holde, K. E. (1995), *Biochemistry*, Benjamin Cummings, Menol Park, Ca.

Matthews, B. W. (1975), 'Comparison of the predicted and observed secondary structure of t4 phage lysozyme', *Biochim Biophys Acta* **405**(2), 442–51.

Maxam, A. M. & Gilbert, W. (1977), 'A new method for sequencing dna', *Proc Natl Acad Sci U S A* **74**(2), 560–4.

McCarthy, F. M., Wang, N., Magee, G. B., Nanduri, B., Lawrence, M. L., Camon, E. B., Barrell, D. G., Hill, D. P., Dolan, M. E., Williams, W. P., Luthe, D. S., Bridges, S. M. & Burgess, S. C. (2006), 'Agbase: a functional genomics resource for agriculture', *BMC genomics* **7**(1), 229.

Menzies, T., Greenwald, J. & Frank, A. (2007), 'Data mining static code attributes to learn defect predictors', *Software Engineering, IEEE Transactions on* **33**(1), 2–13.

Miska, E. A., Alvarez-Saavedra, E., Abbott, A. L., Lau, N. C., Hellman, A. B., McGonagle, S. M., Bartel, D. P., Ambros, V. R. & Horvitz, H. R. (2007), 'Most caenorhabditis elegans microrna are individually not essential for development or viability', *PLoS genetics* **3**(12), 2395–2403.

Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L. & Wold, B. (2008), 'Mapping and quantifying mammalian transcriptomes by rna-seq', *Nat Methods* **5**(7), 621–8.

Murchison, E. P., Stein, P., Xuan, Z., Pan, H., Zhang, M. Q., Schultz, R. M. & Hannon, G. J. (2007), 'Critical roles for dicer in the female germline', *Genes Dev* **21**(6), 682–93.

Nag, A., King, S. & Jack, T. (2009), 'mir319a targeting of tcp4 is critical for petal growth and development in arabidopsis', *Proceedings of the National Academy of Sciences of the United States of America* **106**(52), 22534–22539.

Nam, J. W., Kim, J., Kim, S. K. & Zhang, B. T. (2006), 'Promir ii: a web server for the probabilistic prediction of clustered, nonclustered, conserved and nonconserved microrna', *Nucleic Acids Res* **34**(Web Server issue), W455–8.

Navarro, L., Dunoyer, P., Jay, F., Arnold, B., Dharmasiri, N., Estelle, M., Voinnet, O. & Jones, J. D. (2006), 'A plant mirna contributes to antibacterial resistance by repressing auxin signaling', *Science* **312**(5772), 436–9.

Nelson, J. A. (2007), 'Small rnas and large dna viruses', *N Engl J Med* **357**(25), 2630–2.

Notredame, C., Holm, L. & Higgins, D. G. (1998), 'Coffee: an objective function for multiple sequence alignments', *Bioinformatics* **14**(5), 407–22.

Oinn, T., Li, P., Kell, D., Goble, C., Goderis, A., Greenwood, M., Hull, D., Stevens, R., Turi, D. & Zhao, J. (2007), 'Taverna/my grid: Aligning a workflow system with the life sciences community', *Workflows for e-Science* pp. 300–319.

Ondov, B. D., Varadarajan, A., Passalacqua, K. D. & Bergman, N. H. (2008), 'Efficient mapping of applied biosystems solid sequence data to a reference genome for functional genomic applications', *Bioinformatics* **24**(23), 2776–7.

Palatnik, J. F., Allen, E., Wu, X., Schommer, C., Schwab, R., Carrington, J. C. & Weigel, D. (2003), 'Control of leaf morphogenesis by microrna', *Nature* **425**(6955), 257–63.

Paranjape, T., Slack, F. J. & Weidhaas, J. B. (2009), 'Microrna: tools for cancer diagnostics', *Gut* **58**(11), 1546–54.

Parisien, M. & Major, F. (2008), 'The mc-fold and mc-sym pipeline infers rna structure from sequence data', *Nature* **452**(7183), 51–5.

Pellicer, J., FAY, M. & LEITCH, I. (2010), 'The largest eukaryotic genome of them all?', *Botanical Journal of the Linnean Society* **164**(1), 10–15.

Pertea, G., Huang, X., Liang, F., Antonescu, V., Sultana, R., Karamycheva, S., Lee, Y., White, J., Cheung, F., Parvizi, B., Tsai, J. & Quackenbush, J. (2003), 'Tigr gene indices clustering tools (tgicl): a software system for fast clustering of large est datasets', *Bioinformatics* **19**(5), 651–2.

Pfeffer, S., Sewer, A., Lagos-Quintana, M., Sheridan, R., Sander, C., Grasser, F. A., van Dyk, L. F., Ho, C. K., Shuman, S., Chien, M., Russo, J. J., Ju, J., Randall, G., Lindenbach, B. D., Rice, C. M., Simon, V., Ho, D. D., Zavolan, M. & Tuschl, T. (2005), 'Identification of micrornas of the herpesvirus family', *Nat Methods* **2**(4), 269–76.

Rasmussen, K. R., Stoye, J. & Myers, E. W. (2006), 'Efficient q-gram filters for finding all ε-matches over a given length', *Journal of Computational Biology* **13**(2), 296–308.

Rehmsmeier, M., Steffen, P., Hochsmann, M. & Giegerich, R. (2004), 'Fast and effective prediction of microrna/target duplexes', *Rna* **10**(10), 1507–17.

Ribeiro-dos Santos, A., Khayat, A. S., Silva, A., Alencar, D. O., Lobato, J., Luz, L., Pinheiro, D. G., Varuzza, L., Assumpcao, M., Assumpcao, P., Santos, S., Zanette, D. L., Silva, W. A., J., Burbano, R. & Darnet, S. (2010), 'Ultra-deep sequencing reveals the microrna expression pattern of the human stomach', *PloS one* **5**(10), e13205.

Ritchie, W., Theodule, F. X. & Gautheret, D. (2008), 'Mireval: a web tool for simple microrna prediction in genome sequences', *Bioinformatics* **24**(11), 1394–6.

Rivas, E. & Eddy, S. (2001), 'Noncoding rna gene detection using comparative sequence analysis', *BMC bioinformatics* **2**(1), 8.

Rognes, T. & Seeberg, E. (2000), 'Six-fold speed-up of smith-waterman sequence database searches using parallel processing on common microprocessors', *Bioinformatics* **16**(8), 699–706.

Romeuf, I., Tessier, D., Dardevet, M., Branlard, G., Charmet, G. & Ravel, C. (2010), 'wdbtf: an integrated database resource for studying wheat transcription factor families', *BMC genomics* **11**(1), 185.

Ronaghi, M. (2001), 'Pyrosequencing sheds light on dna sequencing', *Genome Res* **11**(1), 3–11.

Ruan, K., Fang, X. & Ouyang, G. (2009), 'Micrornas: novel regulators in the hallmarks of human cancer', *Cancer Lett* **285**(2), 116–26.

Rumble, S. M., Lacroute, P., Dalca, A. V., Fiume, M., Sidow, A. & Brudno, M. (2009), 'Shrimp: accurate mapping of short color-space reads', *PLoS Comput Biol* **5**(5), e1000386.

Ruvkun, G. (2001), 'Molecular biology. glimpses of a tiny rna world', *Science* **294**(5543), 797–9.

Sanger, F., Nicklen, S. & Coulson, A. R. (1977), 'Dna sequencing with chain-terminating inhibitors', *Proc Natl Acad Sci U S A* **74**(12), 5463–7.

Saqib, M., Zorb, C. & Schubert, S. (2008), 'Silicon-mediated improvement in the salt resistance of wheat (triticum aestivum) results from increased sodium exclusion and resistance to oxidative stress', *Functional plant biology* **35**(7), 633–639.

Sarhan, F., Ouellet, F. & Vazquez-Tello, A. (2006), 'The wheat wcs120 gene family. a useful model to understand the molecular genetics of freezing tolerance in cereals', *Physiologia Plantarum* **101**(2), 439–445.

Sarnow, P., Jopling, C. L., Norman, K. L., Schutz, S. & Wehner, K. A. (2006), 'Micrornas: expression, avoidance and subversion by vertebrate viruses', *Nat Rev Microbiol* **4**(9), 651–9.

Schatz, M. C. (2010), 'The missing graphical user interface for genomics', *Genome Biol* **11**(8), 128.

Schulte, J. H., Marschall, T., Martin, M., Rosenstiel, P., Mestdagh, P., Schlierf, S., Thor, T., Vandesompele, J., Eggert, A., Schreiber, S., Rahmann, S. & Schramm, A. (2010), 'Deep sequencing reveals differential expression of micrornas in favorable versus unfavorable neuroblastoma', *Nucleic Acids Res* **38**(17), 5919–28.

Sethupathy, P., Corda, B. & Hatzigeorgiou, A. G. (2006), 'Tarbase: A comprehensive database of experimentally supported animal microrna targets', *Rna* **12**(2), 192–7.

Seto, A. G., Kingston, R. E. & Lau, N. C. (2007), 'The coming of age for piwi proteins', *Mol Cell* **26**(5), 603–9.

Shendure, J. & Ji, H. (2008), 'Next-generation dna sequencing', *Nat Biotechnol* **26**(10), 1135–45.

Smit, A., Hubley, R. & Green, P. (1996), 'Repeatmasker open-3.0', *www.repeatmasker.org* .

Smith, T. F. & Waterman, M. S. (1981), 'Identification of common molecular subsequences', *J Mol Biol* **147**(1), 195–7.

Song, L., Axtell, M. J. & Fedoroff, N. V. (2010), 'Rna secondary structural determinants of mirna precursor processing in arabidopsis', *Curr Biol* **20**(1), 37–41.

Stevens, R. D., Robinson, A. J. & Goble, C. A. (2003), 'mygrid: personalised bioinformatics on the information grid', *Bioinformatics* **19 Suppl 1**(suppl 1), i302–4.

Stevens, R., Zhoo, J. & Goble, C. (2007), 'Using provenance to manage knowledge of in silico experiments', *Briefings in Bioinformatics* **8**(3), 183–194.

Suh, M. R., Lee, Y., Kim, J. Y., Kim, S. K., Moon, S. H., Lee, J. Y., Cha, K. Y., Chung, H. M., Yoon, H. S., Moon, S. Y., Kim, V. N. & Kim, K. S. (2004), 'Human embryonic stem cells express a unique set of micrornas', *Dev Biol* **270**(2), 488–98.

Sunkar, R., Kapoor, A. & Zhu, J. (2006), 'Posttranscriptional induction of two cu/zn superoxide dismutase genes in arabidopsis is mediated by downregulation of mir398 and important for oxidative stress tolerance', *The Plant Cell Online* **18**(8), 2051.

Sunkar, R., Zhou, X., Zheng, Y., Zhang, W. & Zhu, J. K. (2008), 'Identification of novel and candidate mirnas in rice by high throughput sequencing', *BMC Plant Biol* **8**(1), 25.

Swami, M. (2010), 'Small rnas an epigenetic silencing influence', *Nature Reviews Genetics* **11**(3), 172–172.

Szittya, G., Moxon, S., Santos, D. M., Jing, R., Fevereiro, M. P., Moulton, V. & Dalmay, T. (2008), 'High-throughput sequencing of medicago truncatula short rnas identifies eight new mirna families', *BMC genomics* **9**(1), 593.

Takamizawa, J., Konishi, H., Yanagisawa, K., Tomida, S., Osada, H., Endoh, H., Harano, T., Yatabe, Y., Nagino, M., Nimura, Y., Mitsudomi, T. & Takahashi, T. (2004), 'Reduced expression of the let-7 micrornas in human lung cancers in association with shortened postoperative survival', *Cancer Res* **64**(11), 3753–6.

Tang, F., Barbacioru, C., Wang, Y., Nordman, E., Lee, C., Xu, N., Wang, X., Bodeau, J., Tuch, B. B., Siddiqui, A., Lao, K. & Surani, M. A. (2009), 'mrna-seq whole-transcriptome analysis of a single cell', *Nat Methods* **6**(5), 377–82.

Taylor, I., Shields, M., Wang, I. & Harrison, A. (2007), 'The triana workflow environment: Architecture and applications', *Workflows for e-Science* pp. 320–339.

Teune, J. H. & Steger, G. (2010), 'Novomir: De novo prediction of microrna-coding regions in a single plant-genome', *J Nucleic Acids* **2010**, 10.

Trapnell, C., Pachter, L. & Salzberg, S. L. (2009), 'Tophat: discovering splice junctions with rna-seq', *Bioinformatics* **25**(9), 1105–11.

Trapnell, C. & Salzberg, S. L. (2009), 'How to map billions of short reads onto genomes', *Nature biotechnology* **27**(5), 455–457.

Unver, T., Namuth-Covert, D. M. & Budak, H. (2009), 'Review of current methodological approaches for characterizing micrornas in plants', *Int J Plant Genomics* **2009**(1), 1–11.

Van Rooij, E., Sutherland, L., Liu, N., Williams, A., McAnally, J., Gerard, R., Richardson, J. & Olson, E. (2006), 'A signature pattern of stress-responsive micrornas that can evoke cardiac hypertrophy and heart failure', *Proceedings of the National Academy of Sciences* **103**(48), 18255.

Vaucheret, H. & Chupeau, Y. (2012), 'Ingested plant mirnas regulate gene expression in animals', *Cell Res* **22**(1), 3–5.

Volinia, S., Visone, R., Galasso, M., Rossi, E. & Croce, C. M. (2010), 'Identification of microrna activity by targets' reverse expression', *Bioinformatics* **26**(1), 91–7.

Wang, J. W., Czech, B. & Weigel, D. (2009), 'mir156-regulated spl transcription factors define an endogenous flowering pathway in arabidopsis thaliana', *Cell* **138**(4), 738–49.

Wang, J. W., Wang, L. J., Mao, Y. B., Cai, W. J., Xue, H. W. & Chen, X. Y. (2005), 'Control of root cap formation by microrna-targeted auxin response factors in arabidopsis', *Plant Cell* **17**(8), 2204–16.

Wang, X. (2008), 'mirdb: a microrna target prediction and functional annotation database with a wiki interface', *Rna* **14**(6), 1012–7.

Wang, X., Zhang, J., Li, F., Gu, J., He, T., Zhang, X. & Li, Y. (2005), 'Microrna identification based on sequence and structure alignment', *Bioinformatics* **21**(18), 3610–4.

Watson, J. D. & Crick, F. H. (1953), 'Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid', *Nature* **171**(4356), 737–8.

Wei, B., Cai, T., Zhang, R., Li, A., Huo, N., Li, S., Gu, Y. Q., Vogel, J., Jia, J., Qi, Y. & Mao, L. (2009), 'Novel microrna uncovered by deep sequencing of small rna transcriptomes in bread wheat (triticum aestivum l.) and brachypodium distachyon (l.) beauv', *Funct Integr Genomics* **9**(4), 499–511.

Williams, A. H., Liu, N., van Rooij, E. & Olson, E. N. (2009), 'Microrna control of muscle development and disease', *Curr Opin Cell Biol* **21**(3), 461–9.

Williamson, V., Kim, A., Xie, B., McMichael, G., Gao, Y. & Vladimirov, V. (2012), 'Detecting mirnas in deep-sequencing data: a software performance comparison and evaluation', *Briefings in Bioinformatics* .

Woollard, D., Mattmann, C. A., Medvidovic, N. & Gil, Y. (2008), 'Scientific software as workflows: From discovery to distribution', *IEEE SOFTWARE* **25**(4), 37–43.

Wozniak, A. (1997), 'Using video-oriented instructions to speed up sequence comparison', *Computer Applications in the Biosciences* **13**(2), 145–150.

Xiao, C. & Rajewsky, K. (2009), 'Microrna control in the immune system: basic principles', *Cell* **136**(1), 26–36.

Xiao, F., Zuo, Z., Cai, G., Kang, S., Gao, X. & Li, T. (2009), 'mirecords: an integrated resource for microrna-target interactions', *Nucleic Acids Res* **37**(Database issue), D105–10.

Xu, Y., Zhou, X. & Zhang, W. (2008), 'Microrna prediction with a novel ranking algorithm based on random walks', *Bioinformatics* **24**(13), i50–8.

Xue, C. H., Li, F., He, T., Liu, G. P., Li, Y. D. & Zhang, X. G. (2005), 'Classification of real and pseudo microrna precursors using local structure-sequence features and support vector machine', *BMC bioinformatics* **6**(1), 310.

Yang, W., Chendrimada, T., Wang, Q., Higuchi, M., Seeburg, P., Shiekhattar, R. & Nishikura, K. (2005), 'Modulation of microrna processing and expression through rna editing by adar deaminases', *Nature structural & molecular biology* **13**(1), 13–21.

Yao, Y., Guo, G., Ni, Z., Sunkar, R., Du, J., Zhu, J. K. & Sun, Q. (2007), 'Cloning and characterization of micrornas from wheat (triticum aestivum l.)', *Genome Biol* **8**(6), R96.

Zhang, B. H., Wang, Q. L. & Pan, X. P. (2007), 'Micrornas and their regulatory roles in animals and plants', *Journal of cellular physiology* **210**(2), 279–289.

Zhang, B., Pan, X., Cobb, G. P. & Anderson, T. A. (2006), 'Plant microrna: a small regulatory molecule with big impact', *Dev Biol* **289**(1), 3–16.

Zhang, L., Hou, D., Chen, X., Li, D., Zhu, L., Zhang, Y., Li, J., Bian, Z., Liang, X., Cai, X., Yin, Y., Wang, C., Zhang, T., Zhu, D., Zhang, D., Xu, J., Chen, Q., Ba, Y., Liu, J., Wang, Q., Chen, J., Wang, J., Wang, M., Zhang, Q., Zhang, J., Zen, K. & Zhang, C. Y. (2012), 'Exogenous plant mir168a specifically targets mammalian ldlrap1: evidence of cross-kingdom regulation by microrna', *Cell Res* **22**(1), 107–26.

Zhang, Y. (2005), 'miru: an automated plant mirna target prediction server', *Nucleic Acids Res* **33**(Web Server issue), W701–4.

Zhang, Z., Yu, J., Li, D., Zhang, Z., Liu, F., Zhou, X., Wang, T., Ling, Y. & Su, Z. (2010), 'Pmrd: plant microrna database', *Nucleic Acids Res* **38**(Database issue), D806–13.

Zhou, G. K., Kubo, M., Zhong, R., Demura, T. & Ye, Z. H. (2007), 'Overexpression of mir165 affects apical meristem formation, organ polarity establishment and vascular development in arabidopsis', *Plant Cell Physiol* **48**(3), 391–404.

Zuker, M. & Stiegler, P. (1981), 'Optimal computer folding of large rna sequences using thermodynamics and auxiliary information', *Nucleic acids research* **9**(1), 133.

Zwieb, C. & Wower, J. (2000), 'tmrdb (tmrna database)', *Nucleic Acids Res* **28**(1), 169–70.