NNT : 2017SACLX031

Thèse de doctorat de l'Université Paris-Saclay préparée à l'Ecole Polytechnique

Ecole doctorale n°573
Interfaces : approches interdisciplinaires / fondements, applications et innovation
Spécialité de doctorat : Informatique

par

Mme. Alice Héliou

Analyse des séquences génomiques : Identification des ARNs circulaires et calcul de l'information négative

Thèse présentée et soutenue à l'Ecole Polytechnique, le 10 juillet 2017.

Composition du Jury :

M.	Alain Denise	Professeur Université Paris-Sud	(Président du jury)
Mme.	Christine Gaspin	Professeur INRA Toulouse	(Rapporteur)
M.	Gabriele Fici	Professeur associé Université de Palerme (Italie)	(Rapporteur)
M	Thierry Lecroq	Professeur Université de Rouen	(Examinateur)
Mme.	Claire Toffano-Nioche	Chargé de recherche Université Paris Sud	(Examinatrice)
M.	Mikael Salson	Maitre de conférences Université Lille 1	(Examinateur)
Mme.	Mireille Régnier	Professeur Ecole Polytechnique	(Directrice de thèse)
M.	Hubert Becker	Maitre de conférences UPMC	(Co-directeur de thèse)

À mes grands parents,
Pour m'avoir ouvert les yeux
sur la chance que sont les études.

À mes parents,
Pour m'avoir toujours soutenue
et donné confiance en moi,
Merci.

Remerciements

Je tiens dans un premier temps à remercier mes directeurs de thèse, Mireille Régnier et Hubert Becker sans qui rien de tout cela n'aurait été possible.

Christine Gaspin et Gabriele Fici ont accepté d'être rapporteurs de cette thèse, je les remercie sincèrement d'avoir pris le temps de lire attentivement le manuscrit.

Je souhaite également remercier Alain Denise, Thierry Lecroq, Claire Toffano-Nioche et Mikael Salson d'avoir accepté de faire partie de mon jury pour la soutenance.

La première personne qu'il m'est important de remercier est Laurent Mouchard. Il m'a encadré durant mon stage de M2 à Londres, et m'a fait découvrir les structures de compression et d'indexation qui m'ont passionnée. C'est ce qui m'a donné envie de continuer la recherche par une thèse sur l'analyse des séquences génomiques. Il m'a ensuite apporté son aide, sa bienveillance et son soutien tout au long de ma thèse, je lui en suis extrêmement reconnaissante.

Je dois également beaucoup à Solon Pissis avec qui j'ai travaillé dès mon stage de M2. Il n'a eu de cesse de me proposer de nouveaux problèmes pour pousser toujours plus loin nos travaux sur les mots absents minimaux. Inventif et réactif, c'est un plaisir de travailler avec lui.

Laurent et Solon m'ont donné l'occasion de participer à des workshops et des conférences, grâce à eux j'ai fait des rencontres très enrichissantes. En plus d'être très formateur cela m'a permis de constituer mon jury.

Je tiens également à remercier Yann Ponty, pour avoir apporté beaucoup de dynamisme à l'équipe AMIB. Toujours prêt à discuter, surtout s'il y a du café et du chocolat, il m'a permis de mieux comprendre le fonctionnement du laboratoire. Nos discussions m'ont souvent poussée à l'introspection, merci pour ton temps et pour m'avoir apporté deux co-bureaux. Je remercie Hannu Myllykallio et Hubert Becker pour m'avoir permis de participer aux manipulations de biologie. Hubert, Roxane Lestini et Yoann Collien m'ont beaucoup aidée pour apprivoiser ce nouvel environnement, merci pour votre aide et votre patience.

Durant ces 3 années j'ai pu donner des TDs, c'était très important pour moi qui me destinais à l'enseignement. Ces expériences ont été très enrichissantes, merci

à ceux qui m'ont accordé le monitorat. Je remercie également ceux qui avec qui j'ai donné ces TDs, Philippe Chassignet, David Savourey, Jean-Baptiste Bordes, Pierre-Yves Strub et tout particulièrement Steve Oudot pour m'avoir fait confiance et laissé donner un cours en plus des TDs.

Ensuite, je remercie mes co-bureaux qui ont su rendre les journées de travail et notamment celles de rédaction plus agréables, Afaf et Juraj, mais aussi Amélie et Eliott du LIX. Ainsi que Margaux, Thi-Thuy, Yoann, Minh-Son et tous les autres doctorants du LOB, où il règne une ambiance très sympathique.

Mes co-joggeurs du midi ont énormément contribué à structurer mes semaines autour de sorties hebdomadaires. Je remercie donc Mathieu et Steve, qui m'ont fait découvrir la partie Nord du Plateau, ainsi que Elisabeth, Elise, Roxane et Guillaume pour m'avoir permis de m'aérer l'esprit régulièrement.

Enfin, je remercie celles et ceux qui me sont chers pour leur soutien inconditionnel. Ma mère, toujours là pour me prêter une oreille attentive ; Amélie, ma soeur, pour sa bienveillance et notre complicité inébranlable. Mon véritable regret de cette thèse sera de ne pas avoir trouvé de sujet commun pour un papier A.Héliou & A.Héliou.

Pour finir, je remercie Arnaud, mon âme soeur, qui a toujours su me réconforter quand j'en ai eu besoin. Il m'a permis de garder confiance en moi dans les moments difficiles et de rester fidèle à celle que je suis.

Résumé substantiel

Depuis 1977, la classification conventionnelle des êtres du vivant est celle proposée par Carl Woese, basée sur la comparaison des séquences des ARNr 16S, pour les procaryotes et 18S pour les eucaryotes. Cette classification divise le monde du vivant en trois principaux domaines, les Eucaryotes qui possèdent un noyau, les Bactéries et les Archées. Ces deux derniers domaines correspondent à des organismes souvent unicellulaires, qui ne possèdent pas de noyau. Ils sont donc d'apparence proches et, pendant longtemps ils n'ont pas été distingués. Cependant ils ont des caractéristiques bien différentes, les Archées étant proches des Eucaryotes sur certains points, tels que les protéines impliquées dans les mécanismes de transcription et de traduction.

L'ADN est universellement utilisé pour stocker l'information génétique. Il est transcrit en ARNs, qui, lorsqu'ils sont codants, sont eux-mêmes traduits en protéines. Ainsi l'étude des séquences d'ADN et d'ARN est un outil clef dans la comparaison des espèces et la compréhension des fonctions des différentes macromolécules. Le séquençage de l'ADN et de l'ARN a été inventé dans les années 1970, mais ce n'est que dans les années 2000 que des méthodes à haut débit et peu coûteuses ont été développées, les *Next-Generation Sequencing* (NGS). Leur développement a permis d'importantes avancées en biologie et en santé.

Dans la première partie de cette thèse, nous nous sommes intéressés aux ARNs qui ont la particularité d'être circulaires. C'est-à-dire que leurs deux extrémités sont liées l'une à l'autre par une liaison covalente. Les ARNs circulaires ont été observés dans les trois domaines du vivant depuis plus de vingt ans (Cocquerelle et al. 1993). Cependant ils étaient d'abord vus comme un bruit de transcription. Ce n'est que récemment avec le développement des technologies de séquençage à haut débit que leur importante présence a été révélée (Salzman et al. 2012). Etant circulaires ils sont dégradés plus lentement dans la cellule, ils représentent une partie conséquente des ARNs non codants. Cependant il a également été montré que certains d'entre eux sont traduits en protéine chez la mouche (Pamudurti et al. 2017). Bien qu'étant de plus en plus étudiés, la fonction et le mécanisme de circularisation de la plupart de ces ARNs circulaires restent inconnus.

L'équipe d'Hannu Myllykallio a précédemment mis en évidence l'implication de la ligase *Pab1020* dans la circularisation *in vivo* de trois ARNs, l'ARNr 5S

et deux ARNs à boites C/D, le SR4 et SR9, chez l'archée *Pyroccocus Abyssi*. Le sujet principal de cette thèse est de confirmer le rôle de cette ligase, et d'identifier les ARNs circulaires chez cette Archée.

Pour cela quatre expériences de RNA-seq ont été réalisées :

- A1) Immunoprécipitation de l'ARN avec la ligase puis séquençage : cette méthode permet de sélectionner les ARNs qui interagissent avec la ligase.
- A2) Réplication de la première expérience.
- B) Immunoprécipitation de l'ARN avec la ligase suivi d'un traitement RNase R puis séquençage : le traitement RNase R permet de dégrader les ARNs linéaires.
- C) ARNs totaux traités avec la RNase R puis séquençés.

L'identification des ARNs circulaires à partir des données de séquençage n'est en théorie pas très compliquée puisque les lectures qui couvrent la jonction circulaire ne s'alignent pas normalement sur le génome. En effet, au lieu de s'aligner linéairement en un seul match, elles vont s'aligner partiellement en deux fois, et ces matchs sont inversés sur génome (voir Figure 3.10 pour une illustration). Cependant les ARNs circulaires ne représentent qu'une faible proportion des ARNs d'une cellule, de plus, pour les identifier nous ne pouvons compter que sur les lectures qui couvrent leur jonction circulaire. Ces lectures ne représentent en moyenne que 1 à 3% des lectures dans des séquençages d'ARN totaux. Ainsi, même un faible taux d'erreurs de séquençage peut nuire à la détection des ARNs circulaires en créant des faux positifs qu'il sera difficile de distinguer des vrais ARNs.

De nombreux algorithmes ont été développés pour identifier les ARNs circulaires, ils reposent tous sur le repérage des deux alignements partiels et inversés d'une lecture. Cependant ils n'utilisent pas les même algorithmes d'alignement ni les même critères d'identification d'une jonction. En 2016, Hansen et al. a comparé cinq de ces algorithmes sur des données de séquençage humain. Les résultats sont très divergents (voir Figure 3.2), parmi les 5075 ARN identifiés comme circulaires par au moins un algorithme, seuls 854 (16,8%) sont identifiés par les cinq algorithmes. Finalement les auteurs recommandent d'utiliser deux algorithmes circRNA_finder (Westholm et al. 2014) et find_circ (Memczak et al. 2013), pour augmenter la fiabilité des résultats. Les algorithmes développés sont, pour la plupart, complexes car ils sont adaptés aux génomes eucaryotes chez lesquels de l'épissage a lieu. Chez les Archées, l'épissage n'a lieu que chez les ARNs non-codants. Généralement l'intron contient un motif *Bulge-Helix-Bulge* (BHB) qui est reconnu pour la coupure par une endoribonucléase, puis l'intron est circularisé par une ligase, (Salgia et al. 2003). Les ARNs circulaires ont déjà été identifiés à partir des données de séquençage chez une archée, *Sulfolobus*

solfataricus (Danan et al. 2012). Cette étude nous a servi de base de comparaison pour l'analyse de nos données de séquençage chez *Pyroccocus Abyssi*.

Pour analyser les données des quatre expériences de séquençage nous avons choisi d'élaborer notre propre méthode afin de pouvoir tirer profit de la comparaison des résultats et être aussi exhaustifs que possible. Tout comme Danan, nous avons choisi d'utiliser Blastn pour aligner les lectures. Cet algorithme d'alignement est lent par rapport à l'état de l'art, mais il nous permet d'avoir accès facilement à tous les matchs d'une lecture, même les petits matchs locaux. Cependant nous avons ensuite utilisé des critères différents de Danan et al. . Notre principale différence est le fait d'exiger qu'une jonction soit soutenue par la majorité des lectures circulaires qui la recouvre. Cela nous évite d'identifier des jonctions qui se recoupent et dont la signification biologique est incertaine. En effet, nous supposons que si un ARN est très structuré le séquençage peut mal se passer et donner des artefacts qui sont ensuite de faux positifs. Avec nos différents critères de sélection nous avons identifiés 133 jonctions circulaires chez *Pyroccocus Abyssi*. Nous avons ensuite comparé les résultats de nos différentes expériences, celles ayant subi un traitement RNase R devraient présenter en proportion plus de lectures circulaires que linéaires. Seuls 42 des ARNs identifiés présentent un enrichissement en lectures circulaires avec le traitement RNase. Parmi ces 42 jonctions, 38 sont des snoRNAs à boites C/D, 1 provient de l'intron du tRNA-Trp et 3 proviennent de transcrits non-annotés. Ces 42 jonctions sont toutes retrouvées dans les données des expériences A et C lorsqu'elles sont analysées séparément. Cela soutient fortement le rôle de la ligase, puisqu'elle interagit avec les ARNs que l'on identifie comme étant circulaires.

Nous avons ensuite comparé les résultats de notre méthode d'analyse avec ceux de méthodes existantes sur nos données de séquençage et en contrôle sur des données générées aléatoirement. Nos résultats sont assez divergents, mais les principales différences sont explicables. En effet, circRNA_finder est moins strict au niveau des critères de sélection des jonctions et trouve notamment beaucoup de jonctions dans des régions compliquées à analyser telles que les ARN ribosomiques. Au contraire, find_circ a des critères de sélection très restrictifs, en ne considérant par exemple que certains sites connus d'épissage, ce qui l'empêche d'identifier des jonctions pour l'instant inconnues.

Nous n'avons pas de jeux de données sur lesquels évaluer nos taux de faux négatifs et de faux positifs. Cependant nous pensons avoir un taux de faux négatifs assez faible car dans toutes nos comparaisons, notre méthode a reconnu plus de lectures circulaires que les autres algorithmes. Aussi si l'on souhaite améliorer la rapidité de notre méthode d'analyse en utilisant un algorithme d'alignement plus performant, il faudra alors bien choisir les paramètres pour ne pas augmenter les faux négatifs.

Nous avons maintenant comme projet de confirmer *in vivo* l'implication de la ligase *Pab1020* dans la circularisation. Pour cela nous allons comparer les ARNs circulaires que nous identifierons chez *Thermococcus barophilus* sauvage et avec délétion de la protéine homologue à *Pab1020*. Ensuite nous réaliserons des RNA-seq sur d'autres organismes ayant ou non une protéine homologue à la ligase *Pab1020* de *Pyroccocus Abyssi*. Cela nous permettra d'observer si les caractéristiques des ARN circulaires sont liées à la présence de cette famille d'ARN ligase dans d'autres organismes.

Dans la seconde partie, nous nous intéressons à l'information négative des séquences, les mots qui ne sont pas présents. Les mots absents étant trop nombreux pour pouvoir être calculés rapidement, nous ne considérons qu'un sous-ensemble, qui contient toute l'information nécessaire pour retrouver la séquence initiale. Un mot est dit absent d'une séquence lorsqu'il n'apparait pas dans la séquence. Un mot absent est dit minimal lorsqu'il est absent de la séquence mais que tous ses facteurs propres sont présents dans la séquence. Il a été montré (Crochemore et al. 1998) que le nombre de mots absents minimaux est linéaire en la taille de la séquence. Des algorithmes permettant de calculer les mots absents minimaux ont déjà été proposés, mais les implémentations disponibles n'ont pas des complexités satisfaisantes. Soit elles utilisent une structure de données gourmande en espace soit elles ne sont pas linéaires en temps.

Nous commençons par présenter l'algorithme MAW qui calcule les mots absents minimaux, en utilisant la table des suffixes, en temps et espace linéaire. L'idée du calcul se base sur deux parcours de la table des suffixes ainsi que de la transformée de Burrows-Wheeler. Nous expliquons cet algorithmique, présentons un pseudo-code et les résultats obtenus sur différents jeux de données. Les performances sont meilleurs que celles des autres implémentations disponibles, mais nous avons voulu améliorer le temps de calcul en adaptant l'algorithme au calcul parallèle.

Nous avons ainsi conçu l'algorithme pMAW qui calcule les mots absents minimaux dans les mêmes complexités asymptotiques mais en étant en partie parallélisé. Plus la séquence est grande par rapport à l'alphabet plus le calcul peut être divisé en tâches indépendantes. Nous observons un gain de temps de calcul dès deux processeurs, le temps de calcul est divisé par deux avec seize processeurs.

Cependant l'inconvénient principal de ces deux méthodes de calcul est la mémoire interne nécessaire. Celle-ci dépasse les 100G pour le calcul sur le génome humain, cela est donc impossible sans une infrastructure informatique appropriée. Nous nous sommes donc intéressés au calcul en mémoire externe, l'algorithme est modifié afin que le parcours des tables soit fait de façon séquentielle. Les tables sont

ainsi stockées en mémoire externe et lues petit à petit pour faire les calculs en mémoire interne. L'algorithme em-MAW que nous avons proposé permet de calculer les mots absents minimaux du génome humain avec 1G de mémoire interne en seulement deux fois plus de temps que MAW. Cela devient donc faisable sur n'importe quel ordinateur. Les trois implémentations sont disponibles en ligne à `http://github.com/solonas13/maw`.

Les mots absents minimaux se sont révélés être intéressants pour la comparaison de séquences (Crochemore et al. 2016). Ils proposent une alternative aux méthodes de comparaison basées sur l'alignement ou sur la distribution des mots de taille k, k petit. Puisqu'il est possible de comparer des séquences sur la base de mots absents minimaux, il est intéressant de se demander s'il est possible de faire de la recherche d'un motif donné dans une séquence en se basant sur ces mots absents minimaux.

Nous avons montré qu'étant donnés une séquence et un motif, on peut trouver les positions de distance minimale, en terme de mots absents minimaux, entre le motif et la fenêtre de la séquence commençant à cette position. La complexité en temps de cet algorithme est linéaire en la taille de la séquence et en celle du motif, mais la complexité en espace est linéaire en la taille du motif et est indépendante de celle de la séquence. Il s'agit du premier algorithme de recherche de motifs basé sur les mots absents minimaux.

Les algorithmes de recherche de motifs ont habituellement une complexité en temps qui est indépendante de la taille de la séquence (une fois l'étape de pré-calcul réalisée), et une complexité en espace qui est linéaire en la taille de la séquence, ils sont donc plus rapides mais requièrent plus d'espace. Aussi notre algorithme n'est pas adapté à un nombre de motifs importants. On peut supposer qu'il serait intéressant pour chercher le meilleur alignement pour une lecture qui ne s'alignerait pas sur le génome avec un algorithme conventionnel. Dans les travaux à venir nous souhaitons implémenter cet algorithme et observer ses résultats dans différentes situations, notamment les alignements de lectures sur un génome de référence.

Contents

List of Figures

List of Tables

Acronyms

A Adenine. 5, 6, 10, 17

BHB Bulge-Helix-Bulge. 11, 31, 49

BWT Burrows-Wheeler Transform. 24, 84–89, 96, 115, 116

C Cytosine. 5, 6, 10, 17

cDNA complementary DNA. 36, 37

circRNA circular RNA. 12, 13, 27–31, 34, 36, 39, 44, 46–48, 55, 56, 60–62, 142

DAWG Direct Acyclic Word Graph. 23

DCA Data Compression with Antidictionaries. 69

DNA DeoxyriboNucleic Acid. 3, 5–9, 15–19, 21, 32, 33, 36

emPCR emulsion PCR. 18, 19

G Guanine. 5, 6, 10, 17

IO Input Output Operation. 80, 84

iSA Inversed Suffix Array. 78, 85, 86, 97, 99

LCA Lowest Common Ancestor. 141

LCP Longest Common Prefix. 80–84, 89, 97–104, 108–112, 116

MAW Minimal Absent Word. 67, 108–110, 112, 123–135

mRNA messenger RNA. 6, 7, 10, 12, 13, 46

NGS Next Generation Sequencing. 15, 16, 18, 20, 22, 90

OD Optical Density. 57, 58, 60

PCR Polymerase Chain Reaction. 5, 18, 19, 35

RAM Random Access Memory. 30, 80, 84, 121, 141

RIP RNA Immunoprecipitation Protocol. 34, 36, 43, 60

RMQ Range Minimal Query. 141

RNA Ribonucleic Acid. 3–12, 21, 27–39, 44, 46–49, 51, 55, 56, 60, 142

rRNA ribosomal RNA. 3, 8, 9, 34–36, 39, 46, 49, 52, 60

RT-PCR Reverse Transcriptase - Polymerase Chain Reaction. 34

SA Suffix Array. 78, 81, 84–87, 89, 97–103, 108, 109, 111, 112, 115, 116

SBS Sequencing By Synthesis. 19

SMRT Single Molecule Real Time. 21

T Thymine. 5, 6, 17

tRNA transfer RNA. 11, 12, 27, 29, 46

U Uracil. 6, 10

Part I

Introduction

1

Background about Biology

Contents

1.1 Domains of Life

1.1.1 History

During Antiquity, Aristote was distinguishing the "living beings" into four categories: mineral, vegetal, animal and human. The criteria of membership were established to the naked eye. It was not until the 16th century and the invention of the optical microscope that unicellular organisms have been observed for the first time by Antoni Van Leuwenhoek. In the 20th century, the transmission electron microscope allowed the observation of the cellular ultrastructure. The definition of eukaryotes and prokaryotes is established in 1963 by Stanier, Doudoroff and Adelberg, the first ones are the only ones having a nucleus. In 1977, Carl Woese proposed a new classification based on the comparison of the 16S and 18S rRNA sequences [1]. A

ribosome component is usually a good choice for phylogeny because the ribosomes are highly conserved and found in all known forms of life with the same function; they synthesise proteins from messenger RNAs. The ribosomes are divided into two subunits, a large one and a small one, that fit together. The 16S rRNA is a component of the small unit in prokaryotes, the 18S rRNA is its homologue in eukaryotes. This classification includes a new group that separates from the procaryotes and that was named the Archaebacteria. This classification is still the current one, see Figure 1.1 for a detailed phylogenetic tree. The main characteristics of the three domains of Life are synthesised in Table 1.1. Most prokaryotic cells are small and simple in appearance. They live mostly as independent individuals rather than as multicellular organisms and in an enormous variety of environments, hence most species cannot be cultured by standard laboratory techniques. According to one estimate [2], at least 99% of prokariotic species remain to be characterised.

Characteristics	Archaea	Bacteria	Eukarya
Membrane lipids	Ether-linked	Ester-linked lipids	Ester-linked lipids
Predominantly multicellular	No	No	Yes
Cell wall	Yes	Yes	No
Peptidoglycan	Yes	No	No
Membrane-bound organelles	No	No	Yes
Possible survival above 80°C	Yes	Yes	No
Ribosomes	70S	70S	80S
Circular DNA	Yes	Yes	No
Histones	Yes	No	Yes
Transcription factors required	No	Yes	Yes
RNA polymerase	Several	One	Three
Initiator tRNA	Methionine	Formylmethionine	Methionine
Introns in tRNA	Yes	No	Yes

Table 1.1: The main characteristics of the three domains of Life. Adapted from [3]

1.1.2 Archaea: the third domain of life

Archaea were initially viewed as extremophiles living in harsh environments, such as arctic seawater and salt lakes, but they can be found basically everywhere. Some of them, the methanogens, inhabit human and ruminant guts, aiding digestion. Evolutively they are very interesting as they have some characteristics that are

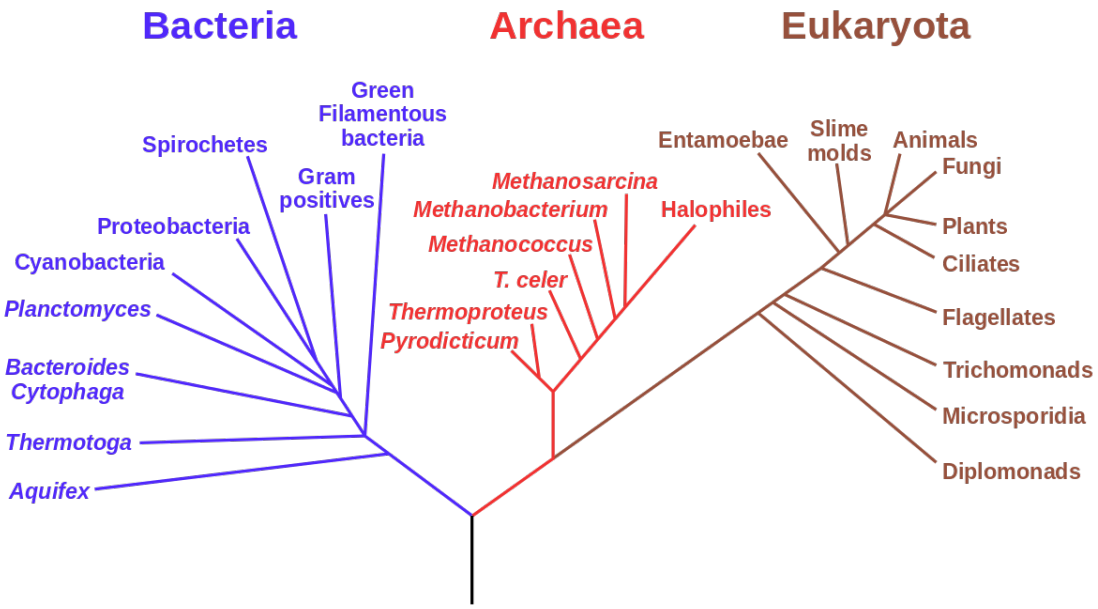


Figure 1.1: Phylogenetic tree based on small-subunit ribosomal RNA sequences showing three domains of life. Taken from [4].

close to bacteria (morphology, reproduction, etc ..) and other ones that are related to eukarya (similarities of proteins, translation and transcription mechanisms, etc ..). Some hyperthermophilic archaea are exploited in biotechnology, for example their DNA polymerases are used for PCR because they are thermostable.

Archaea have been evolving for a long time and they constitute a very heterogeneous phylogenetic group. In our work we focus on the study of *Pyrococcus Abyssi*, an hyperthermophilic archaea that has been isolated from the North Fiji bassin at 2 000 metres, its optimum growth temperature is 96°C.

1.2 What are RNAs ?

1.2.1 From DNA to RNAs

All living cells store their genetic information, necessary to their development and functioning, in the form of macromolecules called DNA. DNA is universal, we can take a piece of DNA from a human cell and insert it into a bacterium, or the reverse, and, under certain circumstances, the information can be successfully interpreted . Most DNA molecules consist of two strands that form a double helix. Each strand is a succession of nucleotides, each of them is composed of a nucleobase - either Adenine (A), Cytosine (C), Guanine (G), or Thymine (T)- and a sugar (deoxyribose) with a phosphate group attached to it.

The nucleotides are joined to one another in a chain by covalent bounds between a sugar and a phosphate group. The sequence of these four nucleotides encodes biological information. The nucleobases of the two strands are bound together (according to pairing rules, A with T and C with G) with hydrogen bonds to make double-stranded DNA.

The stability of DNA helix is determined primarily by hydrogen bonds, but there are two other important forces that maintain the helix structure. The bases, located inside the double helix, interact with each other through the Van der Waals forces. Moreover, adjacent base pairs interact with each other, this kind of interaction is called *stacking interaction*. Stacking interactions are hydrophobic and electrostatic. They depend on the aromaticity of the bases and their dipole moments. There is also the hydrophobic interaction; the nucleobases are hydrophobic whereas the sugar and the phosphate group are hydrophilic. An helical structure tends to reduce the interaction of the bases with the molecules of water. There exist different forms of DNA helical structures, differing in their geometry and dimensions.

The strands run in opposite direction to each other, they are reverse complementary of one another, see Figure 1.2. The bonds between the base pairs are weak compared with the covalent sugar-phosphate links that form the *backbone*. This allows the DNA strands to be pulled apart without breaking. Each strand can serve as a template, for a synthesis of a complementary strand of DNA. This process is called the *DNA replication*. The regulation of DNA replication and its rates in the cell life depend on the type of cells, however the basics are universal, see Figure 1.3.

To function, a cell must do more than copying its DNA, it must also express it. This expression occurs by a mechanism that is the same in all living cells. It begins with the *transcription* in which segments of DNA strand are used as templates for the synthesis of RNAs. Then in some cases it is followed by the *translation*, the synthesis of proteins directed by messenger RNA (mRNA) molecules. In this work we focus exclusively on nucleotidic sequences analysis, thus we will not study proteins and restrain ourselves to DNA and RNA.

The transcription is performed by an enzyme called a RNA polymerase. This enzyme unwinds the DNA helix just ahead of an active site for polymerization, called a *promoter*. Then it moves step-wise along the DNA, exposing a new region of the template strand for complementary base-pairing. A complementary antiparallel RNA chain is extended by one nucleotide at a time in the 5'-to-3' direction. The transcription terminates when reaching a transcription *terminator*.

Like DNA, RNA is a polymer of four types of nucleotides, but one of them is different Uracil (U) replaces Thymine, and the sugar in RNA molecules is a

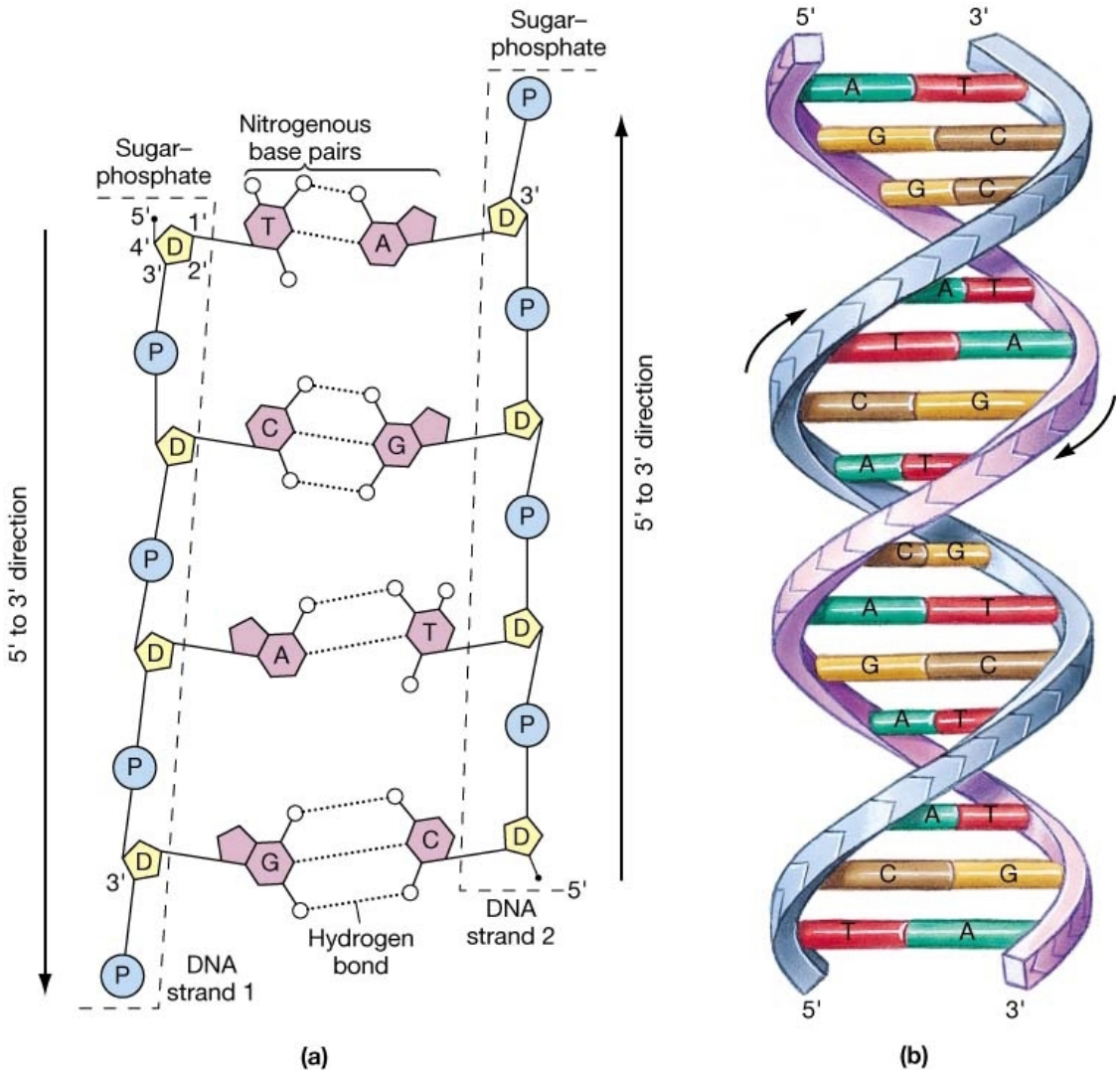


Figure 1.2: DNA structure. A) "Unfolded" view of a double-stranded DNA molecule, showing the two chains of nucleotides, connected in the center by a serie of hydrogen bonds between nitrogenous bases. B) Schematic illustration showing the arrangement of the two strands in the double-helix configuration. The "backbone" on the outside is the sugar-phosphate chain, and the nitrogenous bases form the bridges across the middle. Taken from `http://myhome.sunyocc.edu/~weiskirl/nucleotides_nucleicacid.htm`.

ribose. The segment of DNA that is transcribed into a single strand RNA is called a transcription unit and encapsulates at least one gene. If this gene is coding for a protein, the RNA produced is an mRNA that will be translated into a protein by a ribosome during a process called *translation*. Alternatively, the gene may encode for a non-coding RNA, that will have its function by itself.

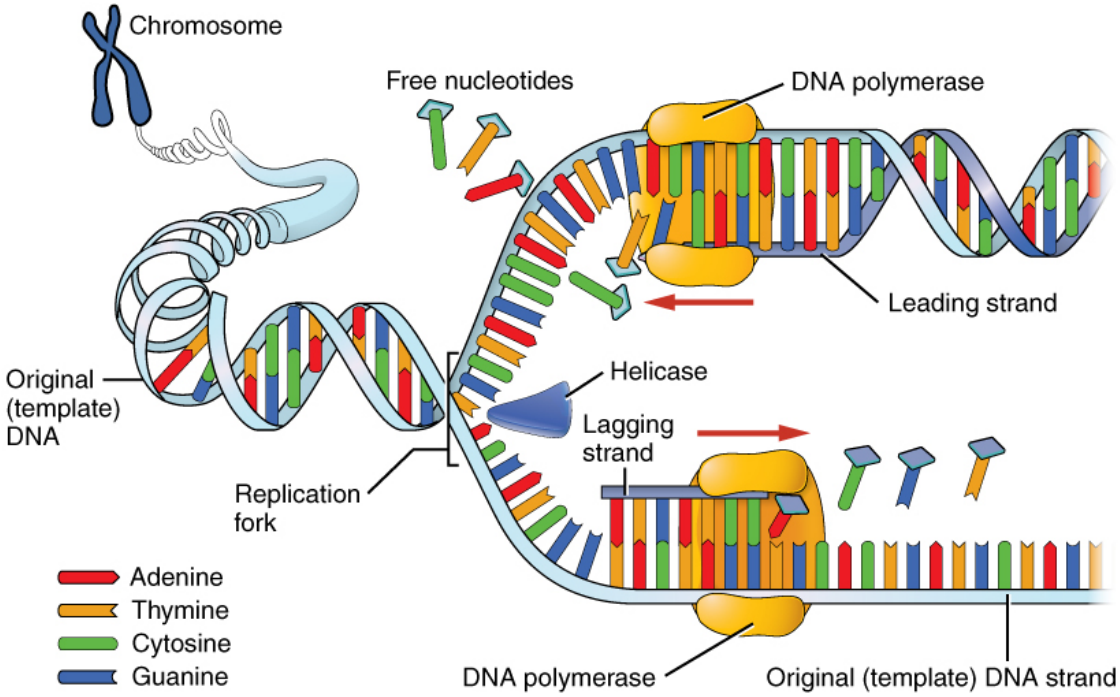


Figure 1.3: DNA replication. DNA replication faithfully duplicates the entire genome of the cell. During DNA replication, a number of different enzymes work together to pull apart the two strands so each strand can be used as a template to synthesize new complementary strands. The two new daughter DNA molecules each contain one pre-existing strand and one newly synthesized strand. Taken from `https://opentextbc.ca/anatomyandphysiology/chapter/3-3-the-nucleus-and-dna-replication/`.

1.2.2 Diversity of non-coding RNAs

Non-coding RNAs are highly abundant, their number within the human genome is unknown, but recent studies suggest the existence of thousands of them. The Table 1.2 summarizes the main categories of non-coding RNAs. Ribosomal RNAs (rRNAs), transfer RNAs (tRNAs) and small nucleolar RNAs (snoRNAs) are the three largest classes of non-coding RNAs. The Lowe lab has been working thoroughly on the tRNAs and snoRNAs, providing web servers for their detection [5]. Those genes are not detected by conventional gene finders that detect protein-coding genes. Indeed they lack or have weakly conserved sequence signals characteristics of protein-coding genes: promoters, terminators, absence of stop codon, poly-A-addition, transcription-factor binding sites etc.... The methods proposed by Lowe et al. combine deterministic search and probabilistic models based on training data from selected species of phylogenetic groups (i.e. mammals, yeasts and archaea). The snoRNAs fall into two major classes: C/D box and H/ACA box RNAs. Most C/D box snoRNAs target particular ribose methylations within rRNA, whereas most H/ACA box RNAs target particular conversions of uridine to pseudouridine within

Type of RNA	Function
mRNAs	Messenger RNAs, code for proteins
rRNAs	Ribosomal RNAs, form the basic structure of the ribosome and catalyse protein synthesis
tRNAs	Transfer RNAs, central to protein synthesis as adaptors between mRNA and amino acids
snRNAs	Small nuclear RNAs, function in a variety of nuclear processes, including the splicing of pre-mRNA
snoRNAs	Small nucleolar RNAs, help to process and chemically modify rRNAs
miRNAs	MicroRNAs, regulate gene expression by blocking translation of specific mRNAs
siRNAs	Small interfering RNAs, turn off gene expression by directing the degradation of selective mRNAs
piRNAs	Piwi-interacting RNAs, bind to piwi proteins and protect the germ line from transposable elements
lncRNAs	Long non-coding RNAs, many of which serve as scaffolds; they regulate diverse cell processes, including X-chromosome inactivation

Table 1.2: Principal Types of RNAs Produced in Cells. Taken from [2].

rRNA. snoRNAs are implied in eukaryote ribosome efficacy. However, as there is no nucleus in Archaea, there can not be any snoRNAs, the Lowe lab introduced the sno-like RNAs that they identified as homologs of snoRNAs genes. Finally, by using a probabilistic model, they identified over 200 sno-like RNAs in seven archaeal genomes [6], among them *Pyroccocus abyssi*.

1.2.3 RNA structure

The functional roles played by non-coding RNAs rely on their interactions with other molecules and thus require the adoption of a specific conformation. Unlike DNA, most RNA molecules are single stranded, they fold on themselves to acquire a compact and functional conformation. Its three-dimensional shape is called *tertiary structure*. The scaffold of this 3D-structure is provided by the *secondary structure*, composed of nucleotides that are bounded together by hydrogen bonds. To entirely characterise the 3D organisation of an RNA molecule we distinguish between three kinds of structure:

- Primary structure: It consists of the sequence of bases read from the 5' extremity to the 3' extremity.

- Secondary structure: It corresponds to a planar representation of the structure. Usually it represents the base pair interactions, the Watson Crick C-G, A-U and the Wooble pair G-U. The pairs create different components in the secondary structure, see Figure 1.4:
  - Helices: they are composed of a stack of base pairs, usually no more that 8 to 10 long.
  - Hairpin loops: they link the 3'- and 5'-ends of a double helix, their length vary from 2 to 14 nucleotides.
  - Interior loops: they separate the helices into two segments by inclusion of residues that are not paired.
  - Bulge loop: they are similar to internal loops, but concern only one strand.
  - Multi-branched loops: they are formed by the insertion of three or more helices.
- Tertiary structure: It is the 3D-localisation of all the atoms of the molecule.

1.2.4 RNA splicing

Canonical splicing

In eukaryotic cells, mRNA transcripts are subject to numerous modifications in the nucleus before they can exit from the nucleus to the cytoplasm. Most eukaryotic reading frames contain *introns*, these parts are transcripted into mRNA but they are removed from the mRNA during a process called the *RNA splicing*, see Figure 1.5. The sequences that are joined together to form the final mature mRNA are called *exons*. Within the intron, a donor site (5' of the intron), a branch site (near the 3' of the intron) and an acceptor site (3' end of the intron) are required for splicing. The splice donor site includes an almost invariant sequence 'GU' at the 5' end of the intron. The splice acceptor site at the 3' end of the intron terminates with an almost invariant 'AG' sequence. Thus the motif 'GU'-'AG' is an almost invariant of canonical splicing. Eukaryotic mRNAs are also modified; by *capping* on the 5' end, *polyadenylation* of the 3' end, methylation and other modifications, see Figure 1.5. The ends modification is necessary to export the mRNA outside the nuclear, to prevent their degradation by exonucleases and to promote the translation.

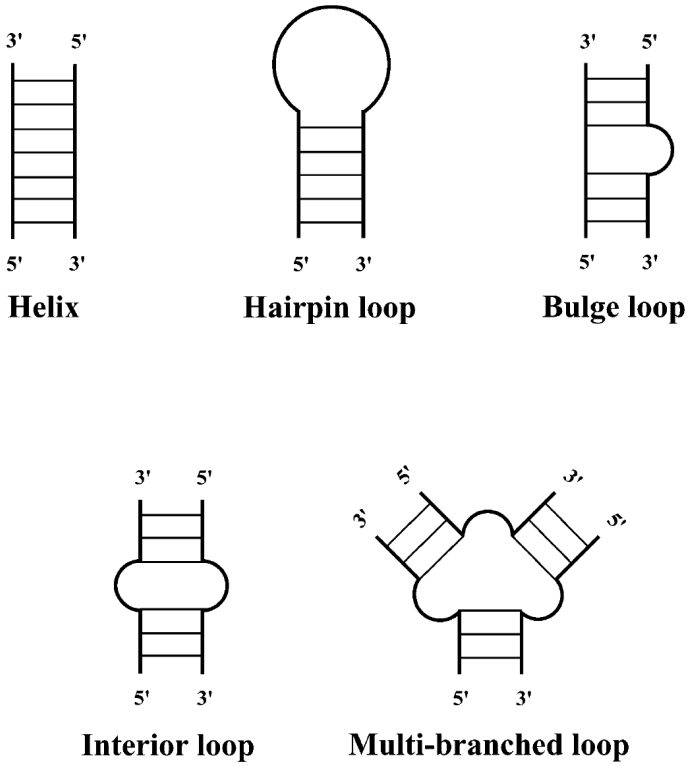


Figure 1.4: Elements of RNA secondary structure: helix, hairpin loop, bulge loop, interior (internal) loop and multi-branched loop. Taken from [7].

tRNA-splicing, back-splicing, trans-splicing and self-splicing

Non-canonical splicing events have been observed in the three domains of life. In Archaea, introns have been found on tRNA and rRNA genes for a long time. At least two protein enzymes, an endonuclease (that cleaves RNA) and a ligase, are known to be involved in RNA splicing in Arachaea. Some archaeal RNA precursors containing introns present a Bulge-Helix-Bulge (BHB) structure that is recognised by the endonuclease as a splicing site [8]. It is interesting to note that archeal RNA splicing endonucleases and eukaryal tRNA splicing endonucleases are homologous, they have similar sequences and are likely to share a common ancestor. Figure 1.6 illustrates a

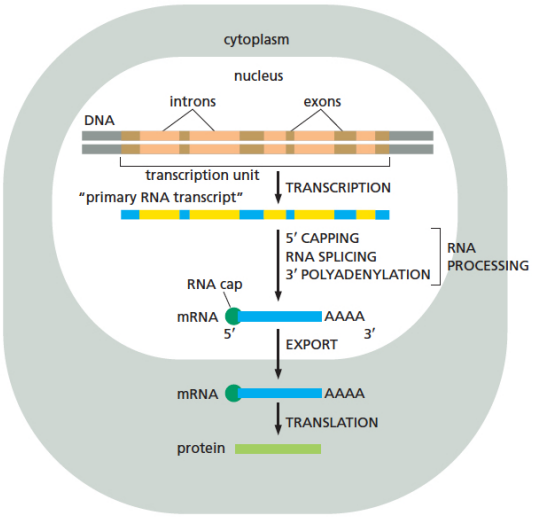


Figure 1.5: In eukatyotic cells, the mRNA resulting from the transcription contains both coding (exon) and non-coding (intron) sequences. This figure represents the different processes undergone by the mRNA before reaching the cytoplasm to be translated into a protein. Taken from [2].

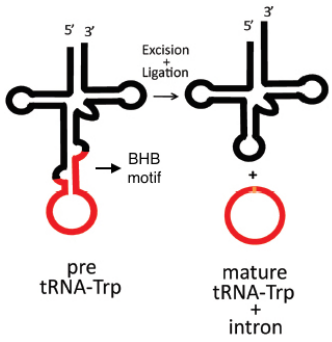


Figure 1.6: Schematic representation of *Sulfolobus solfataricus* tRNA-Trp, which contains a 65 bases intron that is cleaved in the process of tRNA maturation and becomes a stable circRNA. Adapted from [9].

well-known example of a tRNA intron, the *Sulfolobus solfataricus* tryptophan-tRNA (tRNA-Trp). This process generates a tRNA and a circular RNA from the intron.

Back-splicing is a non-canonical splicing event between a splice down and an up-stream splice acceptor (SA) in contrast to a downstream SA in conventional linear splicing, see Figure 1.7. Back-splicing results in circular RNA that are more stable and resistant to RNase R, an exoribonuclease that specifically degrades linear RNA molecules in a 5'-3' direction.

Trans-splicing generates a single RNA transcript from multiple separate precursor mRNAs. There are two categories of trans-splicing events; intragenic, when a mRNA

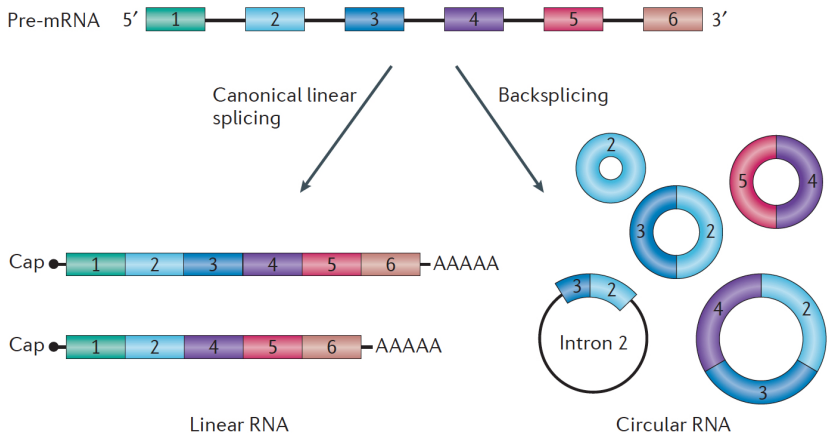


Figure 1.7: Linear RNAs are formed by covalent linkage between an upstream 3' splice site and a downstream 5' splice site of pre-messenger RNA. Whereas circRNA are characterized by a covalent linkage between a downstream 3' splice site and an upstream 5' splice site in a process known as backsplicing. Exons are numbered. Adapted from [10].

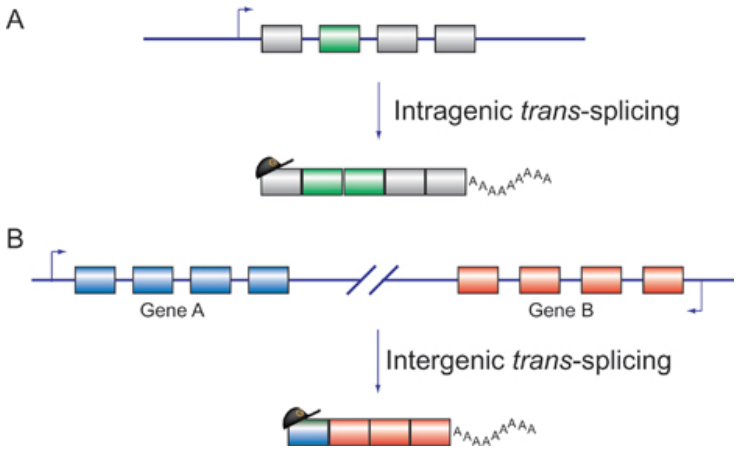


Figure 1.8: A) Intragenic trans-splicing: mRNA containing tandem duplications of specific exons (green) are generated. B) Intergenic trans-splicing: chimeric mRNAs are generated from pre-mRNAs originating from two different genes. Arrows indicate the direction of transcription. Boxes are exons, and horizontal lines are introns. mRNAs have a cap structure at the 5' end and a polyadenylated tail at the 3' end. Adapted from [11].

contains duplicated exon sequences and intergenic, that generate chimeric mRNAs originating from two different genes, see Figure 1.8.

Self-splicing occurs for some introns that are capable of catalyzing their self-splicing reaction. These introns are divided into two main classes: group I and group II, that share different self-splicing mechanisms. More details about self-splicing processes can be found in [12]. Some of these group I and group II introns can form circular molecules, see Table 3.1.

2

Background about computational genetics

Contents

2.1 History

Computational genetics is an interdisciplinary field that uses computational and statistical methods to analyze and interpret genomic data. This field has emerged in the 1970s, it has experienced an explosive growth in the mid-1990s, driven largely by the Human Genome Project and by rapid advances in DNA sequencing technology. The first organism to have its entire genome sequenced was *Haemophilus influenzae* in 1995 of 1,830,140 base pairs of DNA. The first human genome was released in 2001. Nowadays thousand of genomes have been sequenced, and with the sequencing costs declining it now costs below 1,000$ to sequence one human genome. This was made possible by the permanent progress made by the Next Generation Sequencing (NGS) technologies, the availability of huge computer power and storage space.

From a clinical perspective there is great potential for NGS in the management and treatment of human health. Improvements in sequencing technologies has enabled large scale projects, such as the 1000 genomes project whose aim was to map the Human genome common variations [13]. It took them approximatively 5 years to sequence 2,504 individuals from 26 populations and characterised over 88 million variants [14]. Next generation sequencing technologies have also enabled worldwide collaborative efforts in research on cancer, such as the International Cancer Genome Consortium (ICGC) [15] and the Cancer Genome Atlas (TCGA) project [16], whose aim is to catalogue the genomic landscape of thousands of cancer genomes across many disease types.

2.2 Sequencing Technologies

DNA/RNA sequencing is the process of determining the order of nucleic acids of a given fragment. The knowledge of sequences has become essential in numerous fields such as medical diagnosis, biotechnology and forensic biology.

The first DNA sequences were obtained in the early 1970s using laborious methods based on two-dimensional chromatography and DNA labelling [17]. DNA sequencing was invented in 1975 and two different methods have been developed independently, one by Gilbert team and the other by Sanger's. Their principles are different: Sanger's approach is based on a selective enzymatic synthesis while Gilbert's is based on a chemical cleavage technique. Relying on several reviews that have been done on sequencing technologies [18, 19], we present briefly the two approaches and the different resulting technologies that are now widely used.

2.2.1 Sanger sequencing [20]

Sanger method used DNA polymerase to perform a primer extension by incorporating nucleotides coupled with radio-labelled nucleotides lacking a 3'-OH group, dideoxynucleotides, whose incorporation ends the extension of DNA.

Four separate sequencing reactions are performed at the same time. All four contain the standard nucleotides and the DNA polymerase. To each reaction is added only one of the four dideoxynucleotides. The concentration in dideoxynucleotides is approximatively 100-fold lower than the concentration in the corresponding nucleotide, allowing for enough fragments to be produced while still transcribing the complete sequence. Finally, in the reaction containing, for example, a guanine dideoxynucleotide all the fragments end on a G. The resulting DNA fragments are heat denatured and separated by size using gel electrophoresis. The sequence is

inferred from the size of the fragments coming from the four separate reactions. A number of improvements were made to Sanger sequencing in the following years.

2.2.2 Maxam and Gilbert sequencing [21]

Maxam and Gilbert technique differed significantly. Instead of relying on DNA polymerase to generate fragments, radio-labelled DNA is treated with chemicals which break the chain at the specific bases. Similarly to the Sanger method, four reactions are necessary, in each of the reaction the chemical treatments break at a small proportion of one or two of the four nucleotide bases (G, A+G,C, C+T). The guanines (G), the purines (A+G), the cytosines (C), and the pyrimidines (C+T) are modified using specific chemicals. The concentration of the modifying chemicals is controlled to obtain in average one modification per DNA molecule. The modified DNAs is then cleaved by hot piperidine at the position of the modified base. Thus labelled fragments are generated, from the radio-labelled end to the first break in each molecule. They are then denatured and separated by size using gel electrophoresis. The sequence can be inferred from the presence and absence of certain fragments in the different reactions.

At the beginning Maxam-Gilbert and Sanger methods were both quite popular. However Maxam-Gilbert sequencing uses hazardous chemicals and has a technical complexity that prohibits its use in standard molecular biology kits. Thus it was finally the Sanger's method that becomes more popular and that was use to scale-up for high-throughput sequencing.

2.2.3 Next Generation Technology

There are three main companies offering second-generation sequencing platforms: Roche, Illumina and Life Technologies. The Next Generation Sequencing, also called the High Throughput Sequencing enables rapid generation of data by sequencing massive amounts of DNA in parallel.

This parallelisation allowed for an increase of sequencing speed and a reduction of the costs.

The four following technologies are summarised in Table 2.1. They rely upon the same two principles: polymerase-based clonal replication of single DNA molecules spatially separated on a solid support matrix (beads or planar surface) and cyclic sequencing chemistries that allow the detection of nucleotides incorporation.

Technology	Sequencing	Amplification	Run time	Yield (Mb/run)	Read length (bp)	Error rate (%)
Sanger	SBS ddNTP	∅	2 h	0.08	∼1,000	0.1-1
Roche 454	SBS Pyrose-quencing	emPCR	10 h	500	400	1
Illumina HiSeq	SBS RDT	Bridge PCR	6 d	1,000,000	2x125	≥ 0.1
SOLiD	SBL	emPCR	12 d	50,000	35-50	$>$ 0.06
Ion Torrent	SBS H^+	emPCR	7 h	2,000	400	1
SMRT	SBS fluores-cence	∅	0.5-2 h	500	∼10,000	16

Table 2.1: Different NGS technologies characteristic. SBS = Sequencing-by-synthesis; ddNTP = dideoxynucleotide; RDT = Reverse Dye Terminator chemistry; H^+ = Hydrogen ion detection; SBL = Sequencing-by-ligation; SMRT = Single Molecule Sequencing Real-Time; emPCR = emulsion PCR; SE = Single-end read; PE = Paired-end read; Mb = Megabases; bp = base pairs; d = day.

Roche 454 Sequencer (2005)

Roche was the first to enter the market. Its platform distinguishes from the others with longer read lengths, approaching those of Sanger sequencing (700-1000 base pairs (bp)). The sequencing method, the pyrosequencing is different from Sanger's and Gilbert's. It is a bioluminiscent method that measures the release of inorganic pyrophosphate by converting it into visible light using a serie of enzymatic reactions. The sequence is inferred by measuring pyrophosphate production as each nucleotide is added and then washed in turn over the template DNA, fixed to a solid phase. The major difficulty posed by this technique is finding out how many of the same nucleotide there are in a row at a given position. The intensity of light released corresponds to the length of the homopolymer, but the noise produces a non-linear readout above four of five identical nucleotides.

This technology was the first to allow the mass parallelisation of sequencing reactions. Libraries of DNA molecules are first attached to beads via adapter sequences. Then a water-in-oil emulsion PCR (emPCR) is performed to coat each bead in a clonal DNA population. Ideally on average one DNA molecule ends up on one bead, which amplifies in its own droplet in the emulsion, see Figure 2.1a and c. These DNA-coated beads are then washed over a picoliter reaction plate that fits one bead per well. Pyrosequencing then occurs as smaller bead-linked enzymes and nucleotides are washed over the plate, and pyrophosphate release is measured.

Illumina Techonology (2006)

Illumina uses an alternative amplification to the beads by creating DNA cluster directly on the flow cell by bridge PCR, see Figure 2.1b and d. The advantage is that cluster generation by bridge PCR has been fully automated and is therefore more streamlined. The downside is that the success of cluster generation is not known until sequencing has begun, this can result expensive if cluster generation fails. Sequencing itself is achieved in a Sequencing By Synthesis (SBS) manner using fluorescent reversible-terminator nucleotides, which cannot immediately bind further nucleotides as the fluorophore occupies the 3' hydroxyl position. These modified nucleotides and DNA polymerase are washed over the primed, single-stranded flow-cell bound clusters in cycles. At each cycle, the identity of the incorporating nucleotide can be monitored with a sensor by exciting the fluorophores with appropriate lasers, before enzymatic removal of the blocking fluorescent fragments and continuation to the next position. This technology was the first able to produce paired-end (PE) data, in which the sequence at both ends of each DNA cluster is recorded.

Life Technologies - SOLiD 5500 (2007)

Sequencing by oligonucleotide ligation and detection (SOLiD) system differ from the Roche 454 and Illumina because it does not sequence by synthesis but by ligation, using a DNA ligase. Instead of synthesising a complementary strand, they ligate Di-base probes that are fluorescently labelled with 4 dyes. Each dye represent 4 of the 16 possible di-nucleotide sequences. At the beginning of each step a primer of a fixed size is used, then the complementary probe hybridizes to the DNA template and it is ligated by the ligase. Then, after fluorescence is measured the dye is cleaved off and the next complementary di-base probe is ligated. At the end di-nucleotides at regular interval have been translated into a 'colour space'. Next, the synthesised strand is removed, a new primer of a different size is used, and the process is repeated. By combining the result of the different steps, the sequence can be inferred. The read length is limited to 75 base pairs but the two base encoding provides higher accuracy than Illumina sequencing. SOLiD platform is also able to produce paired-end data.

Life Technologies - Ion Torrent

The Ion Torrent uses neither fluorescence nor luminescence. In a manner analogous to Roche 454 sequencing, beads bearing clonal population of DNA fragments, produced via emPCR are washed over a picowell plate, followed by each nucleotide in turn. However, nucleotide incorporation is not measured by pyrophosphate release,

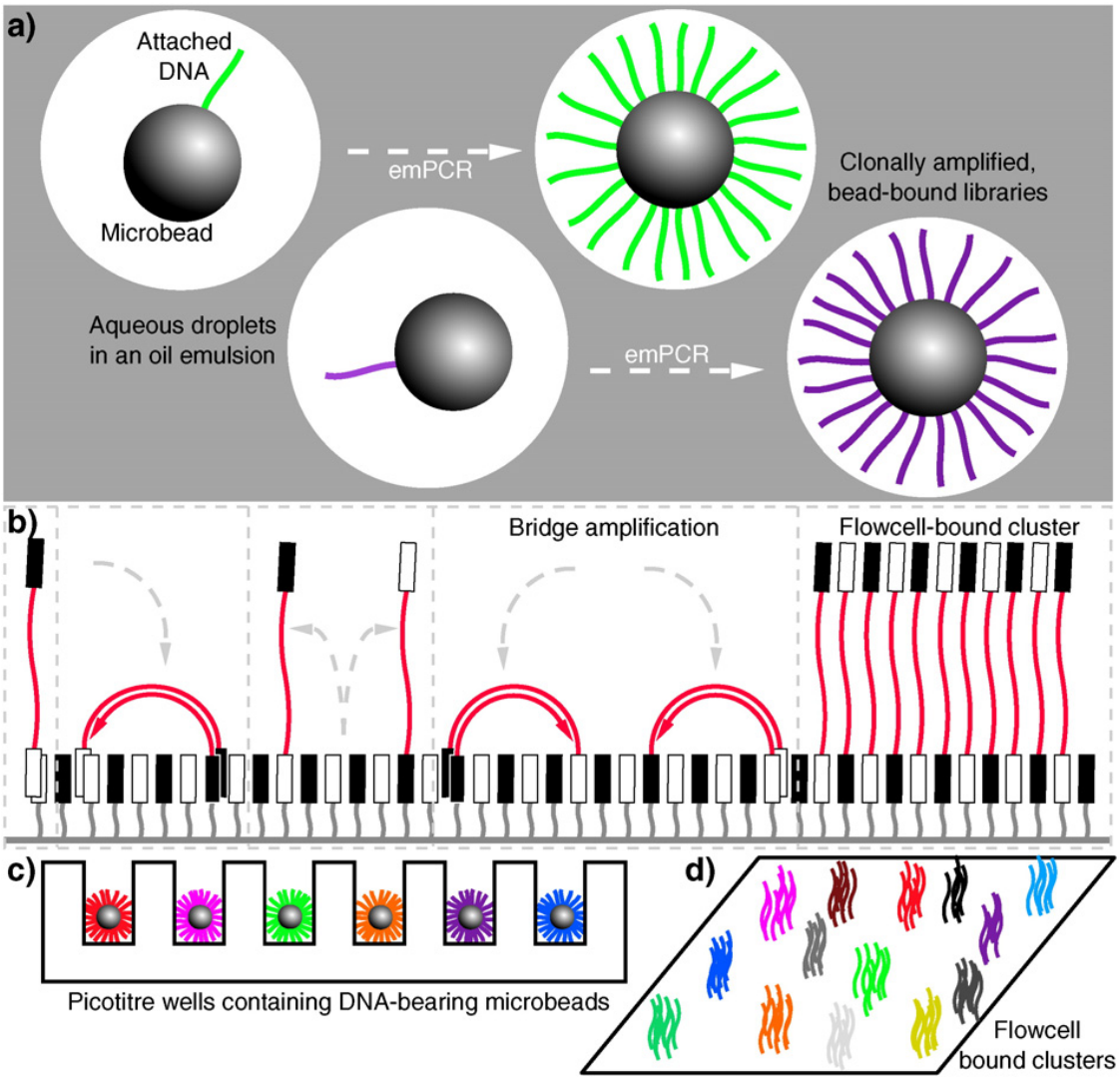


Figure 2.1: NGS parallelized amplification. (a): DNA molecules being clonally amplified in an emulsion PCR (emPCR). This is the conceptual basis underlying sequencing in 454, Ion Torrent sequencing protocols. (b): Bridge amplification to produce clusters of clonal DNA populations in a planar solid-phase PCR reaction, as occurs in Illumina sequencing. (c) and (d) demonstrate how these two different forms of clonally-amplified sequences can then be read in a highly parallelized manner. Adapted from [19].

but with the difference in pH caused by the release of protons (H^+ ions) during polymerisation. This technology allows for very rapid sequencing during the actual detection phase. As Roche 454, due to the loss of signal as multiple matching nucleotides incorporate, it is less able to readily interpret homopolymer sequences.

2.2.4 Third Generation Sequencing Technology

Third generation sequencing technologies work by reading the nucleotide sequence at the single molecule level, they do not require fragmentation nor amplification.

Thus they have the capacity to produce longer reads, and present biaises that are different from those of the second generation technologies. However their error rates are higher, and can be prohibitive for certain applications such as *de novo* genome assembly. Nonetheless, when combined with other sequencing data, like Illumina's, they are usefull for assembly. Moreover, they are convenient for applications tolerant to error rates, such as large structural variant calling.

Pacific Biosystems Real-Time - Single Molecule Sequencing

The Single Molecule Real Time (SMRT), is at this time, the most widely used third-generation technology. During SMRT runs, DNA polymerisation occurs in arrays of microfabricated nanostructures, which are essentially tiny holes in a metallic film covering a chip. In each hole, a single DNA polymerase with a single molecule of a single stranded DNA template is immobilized inside the illuminated region. Incorporated fluorescent nucleotide provides detectable fluorescence that enables the detector to identify the base being incorporated. The dye is then cleaved off by the incorporation, the signal ends as it diffuses out of the illuminated region. PacBio machines are capable of producing incredibly long reads, up to and exceeding 10 kb in length, which are useful for de novo genome assemblies.

Oxford Nanopore Technologies

Single-stranded RNA or DNA can be driven across a lipid bilayer through large ion channels by electrophoresis. Moreover, passage through the channel blocks ion flow, decreasing the current for a length of time characteristic to the nucleic acid. Oxford Nanopore Technologies is the first company offering nano pore sequencers, GridION and MinION, the latter of which is a small, mobile phone size USB device. Despite the admittedly poor quality profiles currently observed, it seems to be a disruptive technology in the DNA sequencing field, producing very long read sequence data far cheaper and faster than was previously possible.

2.3 Data sequencing analysis

2.3.1 *De novo* assembly

De novo assembly is performed when the reference genome for the specie of interest is not available or when one does not want to rely on a existing reference. The aim is to assemble the reads into the full length genome.

Numerous assembly programs, called *assemblers*, have been developed. The main bottleneck to assemble next generation short reads is the repetitive regions.

If the reads are shorter than the repetition it becomes very difficult to detect and locate a repetition. The paired-end sequencing can, to some extent, compensate for read length. With these technologies the two extremities of genomics fragment are processed to produce two reads that are paired, they come from the same fragment. The distribution of the fragments length depends of the sequencing protocol. This information can be usefull when there is an ambiguity for the position on the genome of one of the reads. Several assemblers such as SOAPdenovo [22] and Velvet [23] exploit paired-end sequencing to improve the assembly quality.

The traditional method used for *de novo* assembly, is the overlap graph, introduced by Myers in 1995 [24]. This structure represents each read as a node and overlapping reads are connected by a bi-directed edge. This approach was used for the first assemblies but it is too slow for next-generation data. Indeed the computation of pairwise overlaps has a quadratic complexity in the number of reads.

Another structure based on a similar idea is the De Bruijn graph [25]. A De Bruijn graph is build for a given length k, then every $(k-1)$-mer present in the reads is represented by a node. And every k-mer is represented by an edge that goes from the node corresponding to its longest prefix, to the node corresponding to its longest suffix. Thus for an alphabet of size σ, the De Bruijn graph has at most σ^{k-1} nodes and σ^k edges, independently of the amount of reads. The right choice of k depends on coverage, read length and error rates. Some algorithms, like Velvet recommend to try different k to see which one performs better.

2.3.2 Alignment of sequencing reads

When the reference genome has already been established for the specy of interest, a classical way to analyse NGS data is to find where the reads come with respect to the reference genome.

The true location is not known beforehand and is found by solving an approximate matching problem - that is, searching for occurrences of the read sequence within the reference sequence but allowing for some mismatches and gaps between the two. In general, the most widely used error models are the Hamming distance, which accommodates only for mismatches between the read and a chosen genomic location. And the edit distance, which accounts for mismatches and indels. Other errors models, such as affine gap costs for longer indels, are also possible, often at the expense of a higher computational cost. Algorithms for sequence alignment then produce a BAM file that contains the information about the alignment for each read, position and size of the match, direction of the strand, unique or multiple mapping, number of mismatches and indels, etc...

There is a myriad of sequence alignment algorithms for next-generation sequencing, detailed reviews have been written [26, 27]. We briefly present the different approaches and data structures used for alignment. Most of the algorithms follow the seed-and-extend scheme, where one or more seeds are searched and then extended to cover the whole query sequence. We can divide them in two categories, those based on hash tables and those based on suffix tree or suffix array structures (these two data structures are defined and explained in Chapter 7). An hash table is a data structure that allows an efficient storage of couples (key, value). It also allows a fast access to the data. Here, the keys are usually k-mers and the values are positions of the sequence of interest, usually the reference genome.

The idea of algorithms based on hash tables like SSAHA [28] and SOAP[29] is the same as in BLAST [30]. They consider subsequences of length k, called k-mers. They use an hash table to store for each k-mer of the reference sequence the list of their starting positions. Then they search in the hash table the hits for each k-mer of the read sequence. Finally they try to extend those partial matches into one global match for the read.

The algorithms based on suffix tree or suffix array structures use these index data-structures to do efficient exact pattern matching and then built approximate alignments. In BWA_SW [31] the authors choose to index the reference sequence using a prefix trie and the query with a Direct Acyclic Word Graph (DAWG). Then a dynamic programming is applied by traversing the reference prefix trie and the query DAWG. They use some heuristics to be faster, at each step they only keep the best matches (by default 3). In Bowtie [32, 33] the query is compared to the FM-index (space efficient index data-structure presented in section 7.1) of the reference allowing only a few differences with a backtracking process. In STAR [34], they proceed by first searching for the maximal exact match by the search of the maximum mappable seed in the suffix array of the reference sequence. This search is first performed at the first base of the read, and then it is performed for the unmapped portion of the read. The improvement of this search is that it detects the splice junctions in a single alignment pass. Then in the second phase of the algorithm, STAR builds alignments of the entire read sequence by stitching together all the seeds that were aligned to the genome in the first phase.

Some of these algorithms have been recently improved using general purpose computing on graphics processing units (GPGPU). In 2012 Klus et al. proposed BarraCUDA [35], using the NVIDIA Compute Unified Device Architecture (CUDA) software development environment, they parallelize the BWA's algorithm. In 2013, Luo et al. provided SOAP3-dp [36] that was at the time the fastest, almost 10 times faster than BWA.

2.3.3 Data structures to store raw sequencing data efficiently

Before alignment, the sequencing data produced by the NGS technologies is usually stored using the FastQ format file that contains both the sequence and its corresponding quality scores. After the alignment on a reference genome the reads are stored in SAM/BAM format [37].

For the sake of uniformity, nowadays, when the alignment is not performed by the NGS machine, the reads are still outputted in BAM format but there are qualified as unmapped. This format was developed by the 1000 genomes project [13], it requires around 1 byte (8 bits) per base pair. Recently, the CRAM format has been proposed [38] and adopted by the Global Alliance for Genomes and Health consortium [39]. It is compressed and reference-based, it stores only the differences between the reads and the reference genomes, in average it requires around 0.35 bits per base, much less than the SAM format.

Very recently, they proposed a reference-free full-text index of the sequencing reads from 2,705 individuals [40]. Their data structure is based on a Burrows-Wheeler Transform (BWT) and the FM-index. They managed to store and index 922G reads, representing 87Tbp, into 5.21TB, thus using only 0.5 bit per base. The sequences of the reads are compressed using a BWT, into only 0.46TB, the rest of the space is used to store the metadata (read id, read sample, base quality).

During the summer 2016, I have had the opportunity to work on this project with Thomas Keane team. Our purpose was to achieve a better compression of the metadata. In this aim we proposed another sorting order for the reads, based on their alignment. Thus similarly to the CRAM format, the metadata becomes reference-based for the reads that align successfully, and it stores apart the unmapped reads. On one chromosome for 250M reads, we achieve a compression around 2.3 bytes per read, reducing by a factor of 2.5 the size of the metadata. This method achieves a much better compression for the metadata, but it is based on a non-optimal sorting of the reads for the BWT. Thus the BWT size increases by a factor of 1.5. As a result we estimated the size of the whole index to be around 2.5TB. We did not have the opportunity to verify the scalability of this improvement, because it implies to compute again the BWT of the whole data set, and that step requires numerous computing resources. Thus this work might be incorporated to other impending improvements.

Part II

Identification of circular RNAs

3

Why are circular RNAs particularly interesting ?

Contents

3.1 What are the issues with circular RNAs

3.1.1 General introduction

Circular RNAs (circRNAs) are widely expressed and evolutionary conserved RNAs whose 5'P and 3'OH extremities are covalently linked in a circle. They have been discovered in all domains of life. Some circRNAs are characterized by a non-linear back-splicing event, but others result from lariats or tRNA splicing [9], see Figure 3.1. Although the presence of circRNAs in human cells was established since 1993 by Cocquerelle et al. [42], they were first regarded as transcriptional noise. The prevalence and abundance of these circular RNAs in human cells has

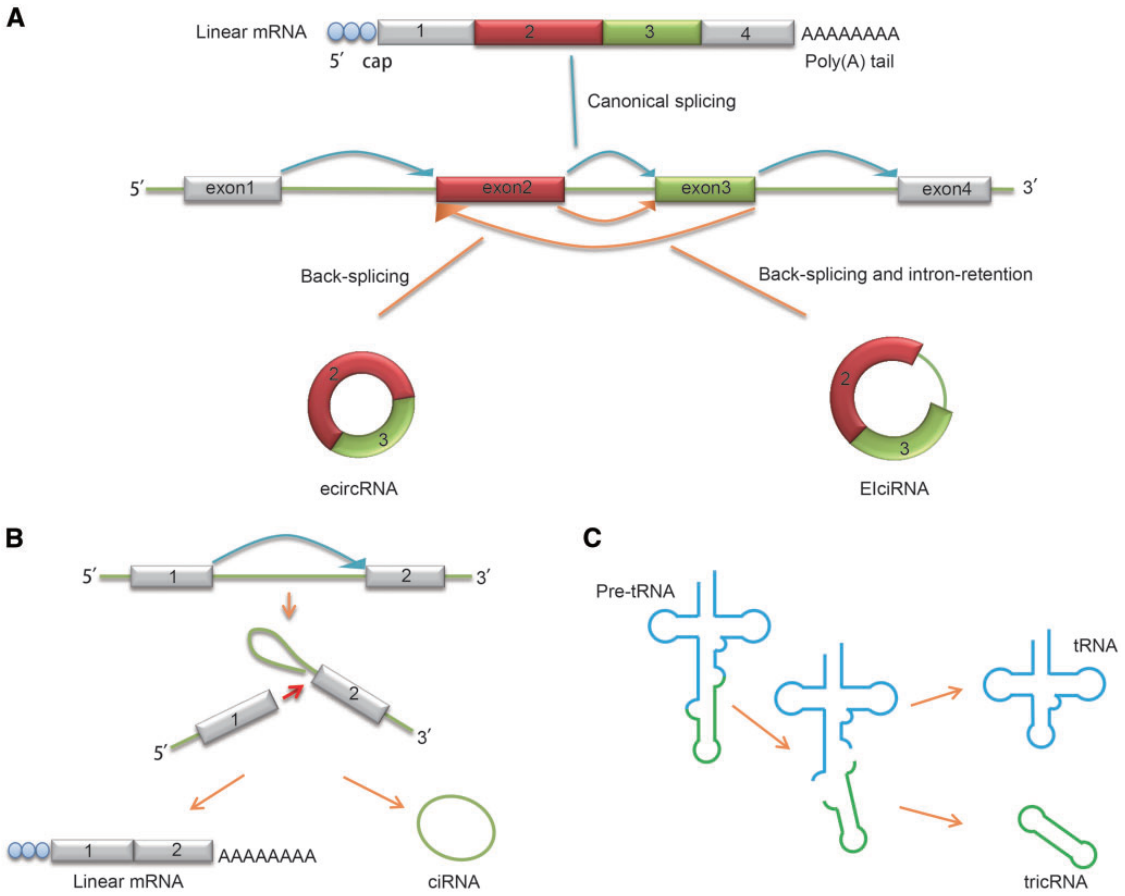


Figure 3.1: Diverse types of circRNAs. (A) Exonic circRNA (ecircRNA) and exon-intron circRNA (EIcircRNA). Canonical splicing produces linear mRNAs while noncanonical splicing (back-splicing) produces ecircRNAs. Particularly, if the intron is retained, an EIciRNA will be generated. (B) Circular intronic RNA (ciRNA). Some lariat introns excised from pre-mRNA by canonical splicing machinery could further form stable ciRNAs. (C) tRNA intronic circRNA (tricRNA). tricRNAs derive from introns that are removed during pre-tRNA splicing. Taken from [41].

only recently been revealed [43–45], with the emergence of high-throughput RNA-seq data analysis. Corresponding experimental validation have proved that they actually represent a class of abundant, stable and ubiquitous RNAs in animals. Although the number of circRNAs identified vary widely among the studies it has become clear that circRNA constitutes an abundant class of non coding RNA. The function of only a few circRNAs has been elucidated some acts a miRNA sponge [44, 46]. MiRNA sponges contain multiple target sites complementary to a miRNA of interest. They neutralise the miRNA and prevent them from blocking the translation of specific mRNAs. Other circRNAs can regulate the function of RNA-binding proteins [47]. Recently Pamudurti et al. [48] provided strong evidence for translation of some circRNAs in fly.

Circularisation enhances circRNA stability, enabling them to be resistant to RNase R exoribonuclease that degrades linear RNA. Their high abundance and evolutionary conservation between species suggest important functions. Genome-wide analyses indicated that the majority of circRNAs are abundant, conserved across species and often exhibit cell-type or tissue specific expression, suggesting potential regulatory roles [49]. Nonetheless, the function of the majority of them remains unknown and there are few models of their mechanism of formation. Jeck et al. [50] and also Lasda and Parker [51] wrote reviews about the diversity of form and function of circRNAs, they summarise numerous previous studies and they distinguished between five general types of RNA described in table 3.1.

Type	RNA circle	Organism	Size
Circular RNA genome	Viroids	Pathogen of plants	250-400 nt
	Hepatitis delta virus (HDV)	Pathogen of humans	1.7 kb
Circular RNA intron	Excised group I introns	Some eukaryotes, some bacterias, some viruses	200-500 nt
	Group II intron circles and intron lariats	Bacteria, some archaea, and some eukaryotic organelles	Up to 3kb
	Circular intronic RNAs (ciRNAs)	Eukaryotes	<200 nt to >3 kb
	Excised tRNA introns	Some archaea	not known
Circular RNA processing intermediate	rRNA precursors	Some archaea	not known
	Permuted tRNAs	Some algae and archaea	not known
Circular noncoding RNA	Some snoRNAs	Some archaea	not known
	RNase P	Some archaea	
Circular RNA spliced exons	Exonic circular RNAs	Eukaryotes	<100nt to > 4kb

Table 3.1: Types and characteristics of circRNA. Adapted from [51].

A repository of circRNAs has been developed, named `circBase` [52], containing all annotation information on circRNAs predicted and identified so far in eukaryotes.

3.1.2 The challenges of identifying circRNAs

Detection of circRNAs depends on reads spanning the back-splicing junction and therefore that map as non linear reads in the genome. The genome-wide discovery of circRNAs is challenging because of biases in both RNA-seq experiments and downstream analysis. Moreover circRNAs constitute a small fraction of reads in common cell lines, around $1-3\%$. Even low sequencing errors can generate false-positive alignments to back-splicing junctions at sufficient levels that they seem to represent truly expressed circRNAs. Then as explained by Szabo and Salzman in their review on the detection of circular RNAs [10], algorithms designed to minimise common sources of false positives can cause systematic 'blind spots' that lead to incorrect results. Quite recently, in 2016, Hansen et al. [53] used common human RNA-seq datasets derived from Hs68 fibroblast, two non-treated (SRR444655 and SRR444975) and two RNase R treated samples, and thus enriched in circRNAs, (SRR444974 and SRR445016) to compare five different detection algorithms, see Table 3.2. Their aim was to evaluate the levels of *bona fide* and false positif circRNAs based on RNase R resistance. The five algorithms rely on the identification of two inverted matches coming from the same read. However they implement distinct alignment methodologies and heuristics, leading to highly divergent results, see Figure 3.2.

In total by merging the five pipelines 5075 unique circRNAs were identified, of these only a modest overlap of 854 circRNAs (16,8%) was observed between all five algorithms.

To assess the level of false positive circRNAs, they compared the number of supporting reads in the RNase R digested samples to the non-treated samples. They required at least 5-fold increase of the number of supporting reads in RNase R treated samples. Based on this criteria they exhibit between 12% (MapSplice and CIRCexplorer) and 28% (CIRI) mis-annotation. 2043 circRNAs were found by only a single algorithm, more than half of them were RNase R sensitive and thus defined as false positives.

The specificities of each algorithm differ in speed, Random Access Memory (RAM) usage, annotation dependance etc ... They also use different aligners and different heuristics, finally their results are dramatically different. Hansen et al. [53] thus suggest combining the predictions of at least two algorithms to obtain better results, they recommend using circRNA_finder and find_circ.

Algorithm	Read type	aligner	junctions considered	Filtering rules	blind spot
MapSplice [54]	PE SE	Bowtie	No restrictions on splice sequence or intron length	Reads: assign single best alignment for each read using a score based on mismatches, base call quality and junction score (based on distribution model)	Junctions where sampling or alignment properties differ from training set used to estimate regression model parameters
find_circ [44]	SE	Bowtie2	GT-AG canonical splicing site and anchors within 100kb	Reads: unique anchor alignment; anchor extension completely aligns read with <3 mismatches Junctions: unambiguous breakpoint	circRNAs comprising small exons or using non-canonical splice signals
Segemehl [55, 56]	PE SE	Segemehl	No restrictions on splice sequence or intron length	Reads: alignment must cover $\geq$ 80% of read	circRNAs lacking high-quality seeds or circRNAs in genes with homologous exons
circExplorer [57]	PE SE	TopHat Fusion	UCSC KnownGene annotated exons within single gene	Reads: Unique alignment	circRNA using unannotated exons or comprising multiples genes
circRNA_finder [58]	PE SE	STAR	Annotated exons within gene or intergenic GT-AG; donor and acceptor within 100kb	Reads:<4 mismatches; unique alignment	circRNAs expressed at moderate to low level
CIRI [59]	PE and SE	BWA-MEM	GT-AG canonical splicing site	Junctions: Filter circRNAs in homologous genes or repeat region and those lacking paired chiastic clipping signal	circRNA using non-canonical splice signals

Table 3.2: circRNA detection algorithms, from [10]. SE: single end, PE: pair ends.

3.1.3 circRNAs in Archaea

Detection of circRNAs in Archaea is supposed to be easiest as splicing events occur only for non coding RNA. In archaea, circRNAs are mainly described as tRNA introns, rRNA introns, rRNA processing intermediates and snoRNA. Usually archaeal introns contain a BHB structure, see Figure 1.6 for an example. The splicing endonucelases recognize the BHB structures for cleaving, then archaeal introns undergo circularisation by an RNA ligase. These mechanisms have been studied from a long time [60] and more recently by Salgia et al. [8] in *Haloferax volcanii*. They showed that in vitro the *H. volcanii* ligase can circularize both endonuclease-cleaved introns, and synthetic substrates. Danan was the first to combine experimental and computation approaches to map circular transcript in an archaea, *Sulfolobus solfataricus* [9]. They compared the results of RNA-seq experiment enriched in circRNAs by a RNase R treatment with a non-treated RNA-seq experiment. They require for a circRNA to be covered by circular reads from both experiments.

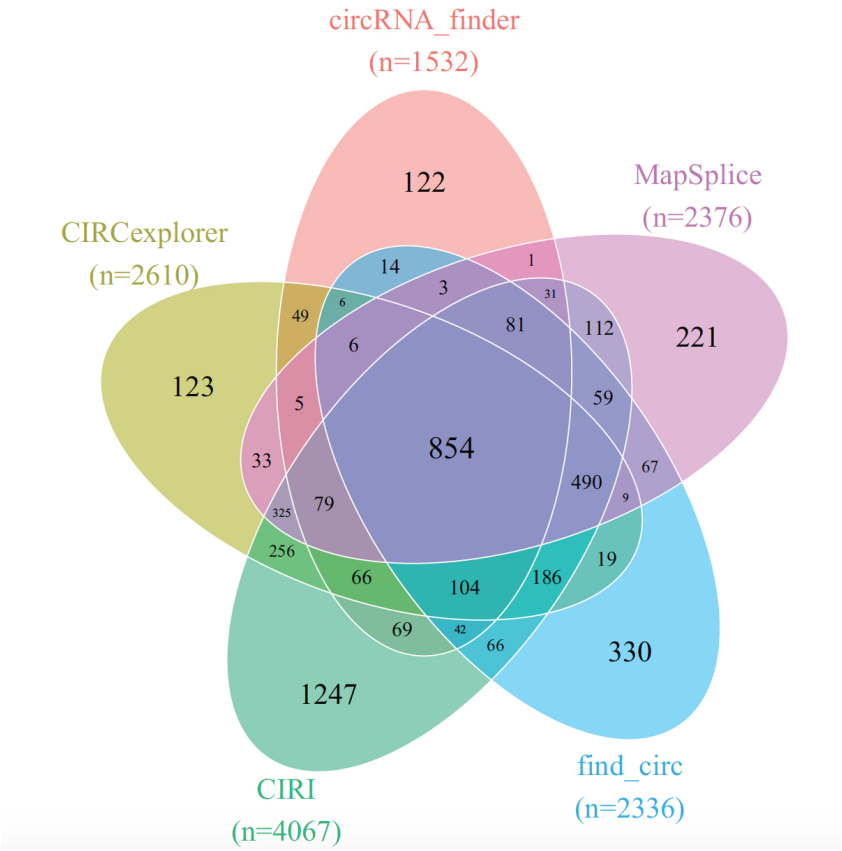


Figure 3.2: Prediction of circRNAs by five different prediction algorithms. Taken from [53].

3.2 Our motivations

3.2.1 Identification of a putative ligase

Our team, the Laboratoire d'Optique et Biosciences, studied the mechanisms of replication and repair of DNA in archaea. Thus they analysed the two open reading frames encoding proteins previously predicted in *Pyrococcus abyssi*, *Pab2002* and *Pab1020*. They showed that while *Pab2002* is indeed a DNA ligase, *Pab1020* has no effect on DNA but has an RNA ligase activity [61]. Therefore, they discovered a putative RNA ligase family (InterPro code IPR001072) whose founding member is the *Pyrococcus abyssi* reading frame, *Pab1020*. A *Pab1020* monomer consists of fours domains: the amino terminal (N-term), the catalytic nucleotide transferase (NTase), the dimerization (Dim) and the carboxy-terminal (C-term) domains.

Biochemical and structural studies [61] had revealed that unlike others DNA and RNA ligases, *Pab1020* displays a homodimeric structure where two monomers are related by non-crystallographic dyad symmetry, see Figure 3.3. Moreover Becker et

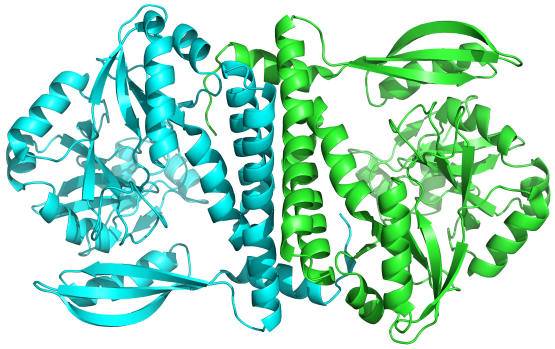


Figure 3.3: The structure of *Pab1020* coloured by chain.

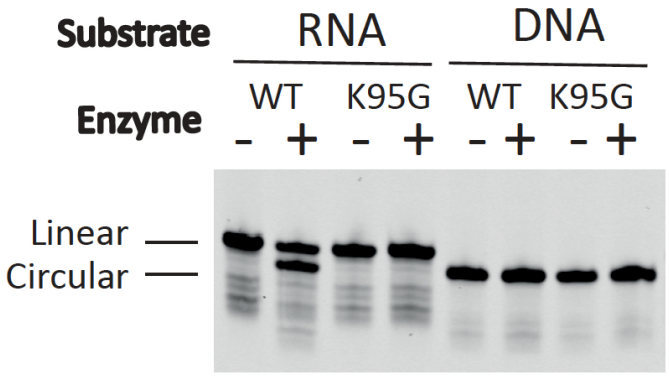


Figure 3.4: RNA and DNA ligation assays with WT and mutant K95G of *Pab1020* RNA ligase. Standard ligation reactions containing 10pmol Cy5-RNA or -DNA molecules (18 mers) and 200 pmol RNA ligase *Pab1020* were incubated 90min at 50°. Reaction products were resolved on denaturing PAGE and a 700nm scan of the gel was performed on Licor Odyssey Infrared Imager. While no activity was observed with DNA substrate, *Pab1020* RNA ligase circularized an RNA oligoribonucleotide as shown on the gel with the apparition of a lower band corresponding to circular RNA molecules. Expectedly, a control reaction with an inactive enzyme (mutant K95G) presented no lower band.

al. [62] showed that *Pab1020* only has a circularization activity on single stranded RNA and no activity on single stranded DNA nor on the different RNA/DNA homo- hetero- oligonucleotide duplexes tested. They did the same experiments with a mutant of *Pab1020* in which the catalytic nucleotide transferase domain was modified by a mutation at position 95, the lysine is mutated into a glycine. In these experiments, with small synthetic RNAs oligonucleotides (18 mers) the circular product migrated 'faster' than the linear substrate, see Figure 3.4. No circular products were observed with the mutant, thus confirming the specific circularization was indeed catalyzed by *Pab1020*.

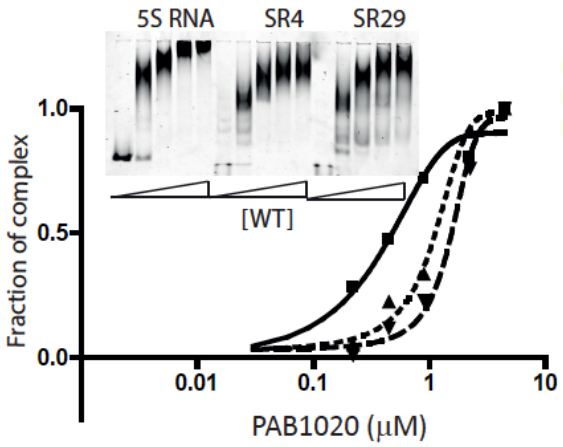


Figure 3.5: RNA binding between *Pab1020* RNA ligase (0.2 to 4.5 μM) and the in vitro transcripts (0.4 μM) corresponding to C/D Box RNAs SR4 (■) and SR29 (▲) and 5S rRNA (▼) was analyzed by EMSA. A fraction of protein-RNA complex formed was plotted as a function of input protein. Insert: On the EMSA gel, the amount of the higher molecular weight bands, corresponding to *Pab1020*-nucleic acid complexes, increased as a function of the protein concentration

3.2.2 *In vivo* identification of circRNAs

Becker et al. have performed RNA Immunoprecipitation Protocol (RIP) assays to obtain RNAs that interact with *Pab1020*, they identified the most abundant RNAs, the 5S, the 16S and the 23S rRNAs. Among them they choose the smaller because it was easier to study. Moreover by relying on Danan [9] study of *Sulfolobus solfataricus* transcriptome, they choose to include in their study two C/D Box RNA (sR4 and sR29). They have assayed the ligation activity of *Pab1020* RNA ligase using the linear fluorescent Cy5-RNA transcripts for three RNAs. Fluorescent RNA substrates were prepared by *in vitro* transcription with T7 RNA polymerase capable of incorporating Cy5-labeled nucleotide analogs. The electrophoretic mobility shift assays (EMSA) indicated that the three transcripts formed specific RNA-protein complexes at near stoichiometric conditions, see Figure 3.5.

To further identify circular RNA molecules, they used RNase R exoribonuclease treatment to discriminate between circular products and linear substrate RNAs. They observed that the fluorescent transcript corresponding to *P. abyssi* 5S rRNA became partially resistant to RNase R treatment after incubation with *Pab1020*, whereas the linear substrate RNA was totally degraded, see Figure 3.6.

They further confirmed the RNA ligation activity of *Pab1020* on the 5S rRNA and the Box C/D RNAs SR4 and SR29 by using inverse RT-PCR on circular RNA matrix. In the inverse RT-PCR, outward facing (inverse) primers are used, this way,

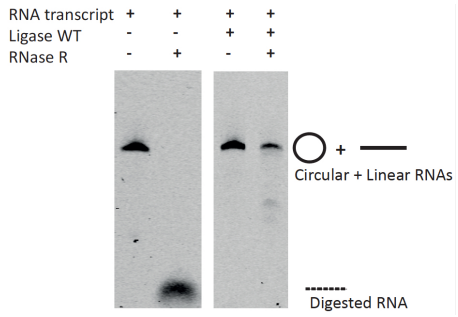


Figure 3.6: *In vitro* transcript of 5S rRNA was incubated (right panel) or not (left panel) with *Pab1020* RNA ligase (WT) for 120min at 55°C. After incubation, recovered RNAs were treated or not with exoribonuclease RNase R for 120min at 37°C before analysis on a 7% acrylamide 8M urea gel.

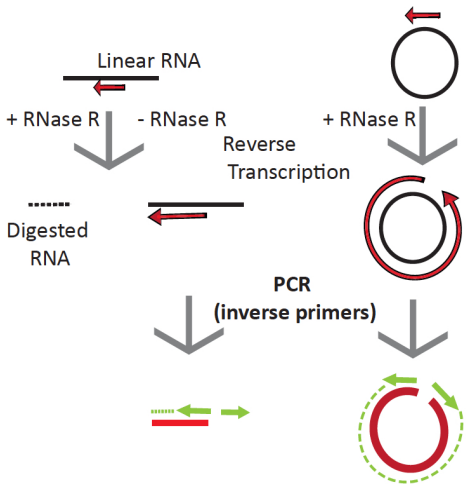


Figure 3.7: Schematic illustration of RT-PCR experiments on linear and circular RNAs with divergent primers to distinguish linear RNAs from circular RNAs products after incubation with *Pab1020* RNA ligase. Only reverse transcription and PCR reactions on a circular RNA template will lead to the total amplification of the substrate sequence.

the reverse transcription is expected to amplify only circular templates, whereas only a small fragment of linear RNA template remains, see an illustration on Figure 3.7. Indeed, they observed a full-length RT-PCR product (indicated by the asterisk in Figure 3.8) for the three selected RNAs. It confirms RNA circularization by *Pab1020*. As negative control, in absence of RNA incubation with *Pab1020* RNA ligase, the full-length amplification products corresponding to circular molecules (5S, SR4, SR29) were not observed (Figure 3.8).

These results from RNase R treatments and inverse PCR amplifications confirmed that the RNA ligase encoded by *Pab1020* gene catalyses the intramolecular ligation of RNA molecules.

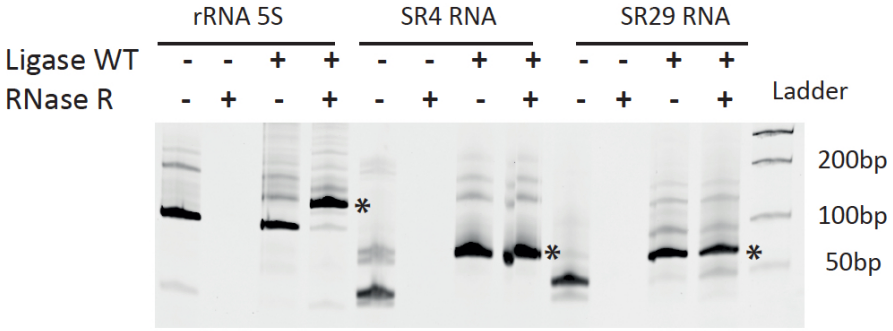


Figure 3.8: complementary DNA (cDNA) generated using outward facing primers on RNAs previously incubated (+) or not (-) with *Pab1020* RNA ligase and in the present (+) or absence (-) of RNase R were separated by gel electrophoresis. A full-length product attesting the amplification of circular RNA molecules, indicated by the asterisk, was observed for 5S rRNA (128bp), Box C/D SR4 RNA (68bp) and Box C/D SR29 RNA (66bp). Circularization was observed only in the presence of *Pab1020* RNA ligase.

3.2.3 RNA-seq experiments

As their biochemical studies confirmed that *Pab1020* acts as RNA ligase, Becker et al [62] further investigated the substrate specificity of this protein in the cell. To do so, four different RNA samples were analyzed using experimental and computational RNA sequencing pipeline, we further explain and justify in chapter 4, the Figure 3.10 gives an overview of the whole process.

Four different RNA-seq experiments were conducted:

A1) Pulldown ligase: RIP assay followed by an RNA-seq,

A2) Pulldown ligase: replicate of experiment A1,

B) Pulldown ligase + RNase R treatment: RIP assay followed by an RNase R treatment to enrich in circRNA before the RNA-seq,

C) Total RNA + RNase R treatment: RNA extraction followed by an RNase R treatment and an RNA-seq.

In the first three experiments (A1, A2 and B) the samples were obtained by co-immunoprecipitation of RNA ligase *Pab1020* after formaldehyde crosslinking between *Pab1020* and cellulars RNAs, RIP assays. In the fourth experiment, total RNA sample was extracted from a stationary phase culture of *P. abyssi*. To enrich the total RNA fraction in circRNAs, an RNase R, that specifically degrades linear RNA molecules in a 3'-5' direction, was used in experiments B and C.

The protocol used for sequencing is Ion Torrent PGM RNA-seq, it is divided into three main steps illustrated in Figure 3.9:

- Fragment the whole transcriptome RNA;
  Partially fragment the RNAs using RNase III enzyme, that cleaves inter- or intramolecular regions of double-stranded RNA [63]. This resulted into formation of RNA fragments that were approximately 100 -150 bp.

- Construct the whole transcriptome library;
  Hybridize and ligate the adaptors on the RNA, then perform reverse transcription to obtain cDNA. The cDNAs obtained are then purified and amplified by PCR.

- Prepare the template and start the run

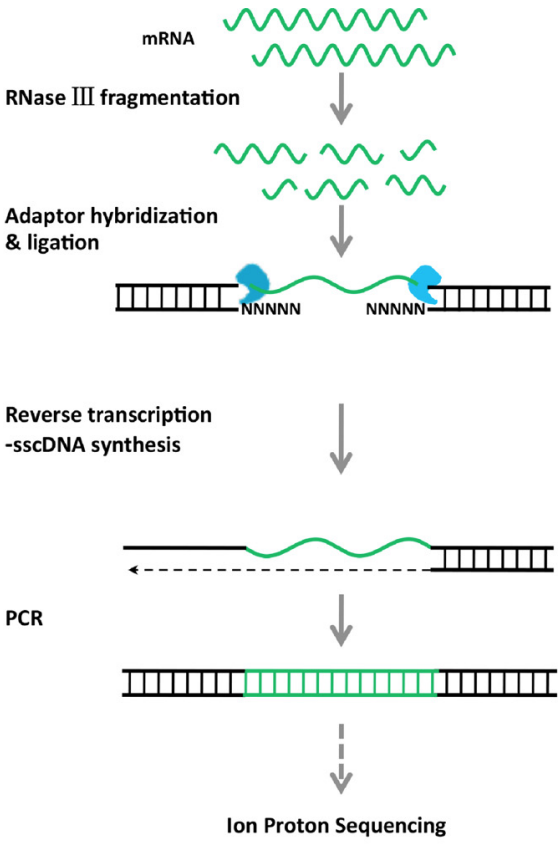


Figure 3.9: Life Technologies' protocol: RNAs were fragmented by RNase III. Then the adaptors hybridize and are ligated on the RNA. cDNAs are obtained with a reverse transcription. These cDNAs are fixed on beads, on average one per bead, then they are amplified by an emulsion PCR. The beads are said to be enriched, because several identical RNA are fixed into the same bead. Finally, the monoclonal beads, those having only identical sequences are then used for sequencing. Adapted from [64]

Typical sequencing runs yielded approximatively to 400 000 reads with a read size of 80 to 90 base pairs. Our aim was then to analyse these data to identify

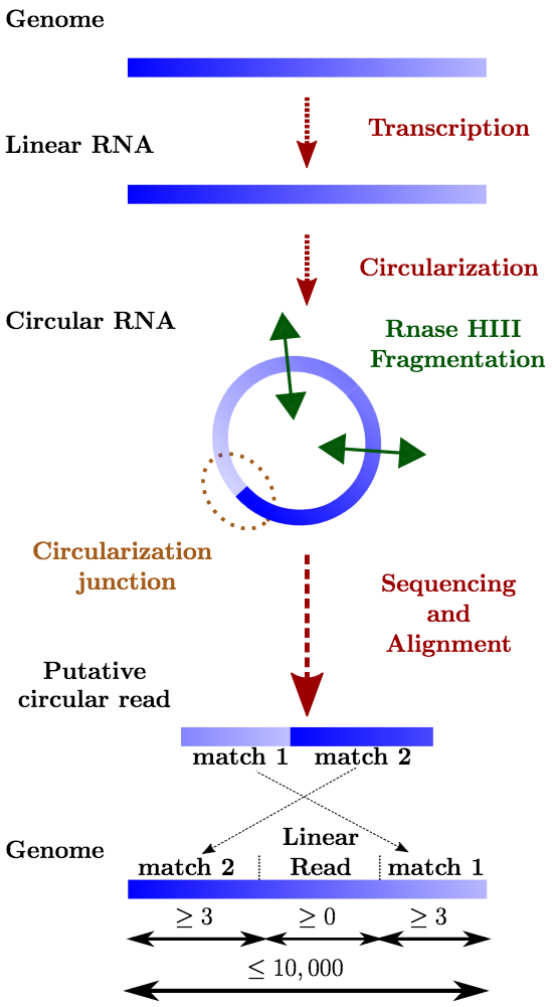


Figure 3.10: The pipeline for identification of circularization junctions using RNA samples isolated from *P. abyssi* cells using IonTorrent semiconductor-based sequencing technology is shown. Identical computational approach was used for the four samples. Obtained linear and circular RNA molecules were fragmented at least once (indicated by a double arrow in green) using RNase III treatment. Following reverse transcription, samples were sequenced and obtained reads were aligned to the *P. abyssi* reference genome using Blastn. Reads were considered circular if two permuted matches covering the whole read were detected.

circular reads. The comparison of the different experiments would then help validate the identification, with the enrichment expected with the RNase R treatment.

4
Data analysis

Contents

We have seen that a lot of algorithms have been developed to identified circular RNAs (see Table 3.2), but their results are dramatically different. Thus, we decided to propose our own pipeline to identify circular RNAs, it does not have to be particularly fast, nor complex. We are studying archaea RNA-seq thus we do not consider linear splicing events in our analysis. The whole pipeline and the different python codes used can be found here `http://www.lix.polytechnique.fr/Labo/Alice.Heliou/circRNA/`. The Figure 4.1 presents an overview of the different steps of our analysis pipeline.

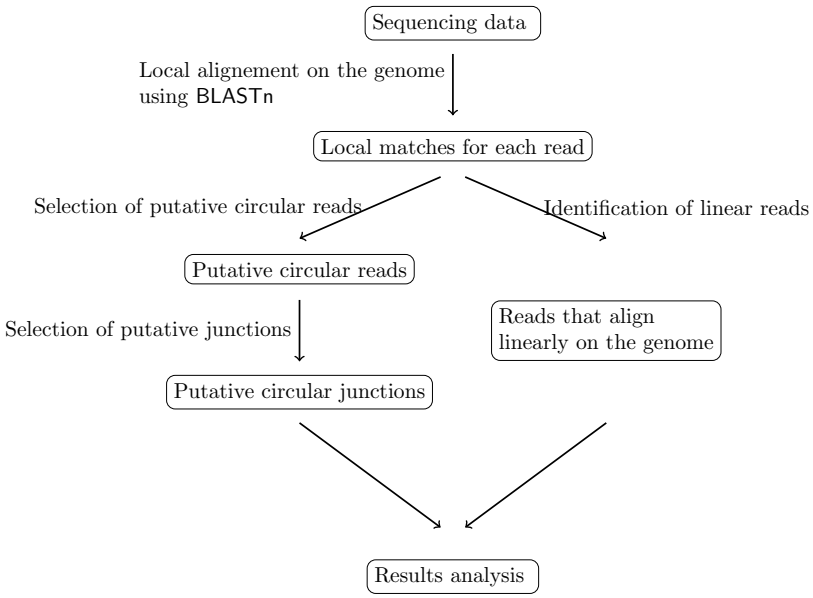


Figure 4.1: Overview of our analysis pipeline

4.1 Reads mapping

4.1.1 Alignment

There are multiple modern algorithms for reads mapping, we have presented the most popular in section 2.3.2. However as we want to be as exhaustive as we can, we decided to use BLAST [30], the oldest one, with very little heuristic. This algorithm is quite slow but it allows us to account for every local match of every reads.

We consider every reads obtained from the Ion Torrent sequencing machine, even those having low quality score and most of all, those that do not match on the genome (reads that cover the circular junction are not expected to align well on the genome). The reference genome used for the alignment is *Pyroccocus abyssi* GE5 (GenBank: NC_000868.1, 1,765,118 base pairs). Read mapping was performed using Blastn (version 2.2.26+) [65] with the Megablast option using the following default parameters:

- word size at least 11,
- gap opening penalty of 5,
- gap extending penalty of 2,
- mismatch penalty of 3 and match reward of 1,

- expect value threshold of 10, this parameter describes the number of hits with the score of the match one can 'expect' to see by chance. The lower the expect value, or the closer it is to zero, the more 'significant' the match is. 10 is the default value, it means that, it is expected to see 10 matches with a similar score simply by chance,
- maximum number of outputs is 250 alignments per query.

The maximum number of allowed outputs was not limiting our analyses, as the highest observed number of alignments for any given query was 182.

The command line to run this alignment step is provided in Code 4.1, four variables have to be set by the user, the number of threads, the output file in xml format, the input file in fasta format and the index of the reference genome.

```
./blastn -task megablast -outfmt 5 -word_size 11 -gapopen 5 -
   gapextend 2 -penalty -3 -num_threads $NB_THREAD -out $DIR$OUTPUT"
   .xml" -query $DIR$OUTPUT".fasta" -db $DIR$REF".db"
```

Code 4.1: Command line for blastn (version 2.2.26+).

4.1.2 Detection of putative circular reads

When analysing BLAST output we distinguish between four cases, the first one is the classical one, the match cover almost entirely the whole read, we say that this read maps linearly. The second and the third cases concern circular reads. To detect putative circular reads in sequencing data, all reads having two matches (from the Blastn output) that together covered the whole read, were selected. We consider only the inverted matches, see Figure 4.2 for an illustration, with no overlap on the reference genome that are located within a 10,000 nucleotides window on the genome sequence. The simplest case is when two matches cover almost entirely the whole read, without overlapping, and have inverted matches on the genome. However there is another case that we need to take into account, is when a match cover a large part of the read but more than two nucleotides and less than eleven (our minimum word size parameter) are missing to cover the read. BLAST was not able to search for this small part in the genome, because it is smaller than the word size parameter. Thus, we look 'naively' for the small match in a window of 10,000 nucleotides such that the two matches will be inverted. If we find a match, then the read is considered as circular. The last case occurs when the read does not map in a linear way nor in a circular way. Then we consider this read as unmapped and we do not take it into account in further analyses.

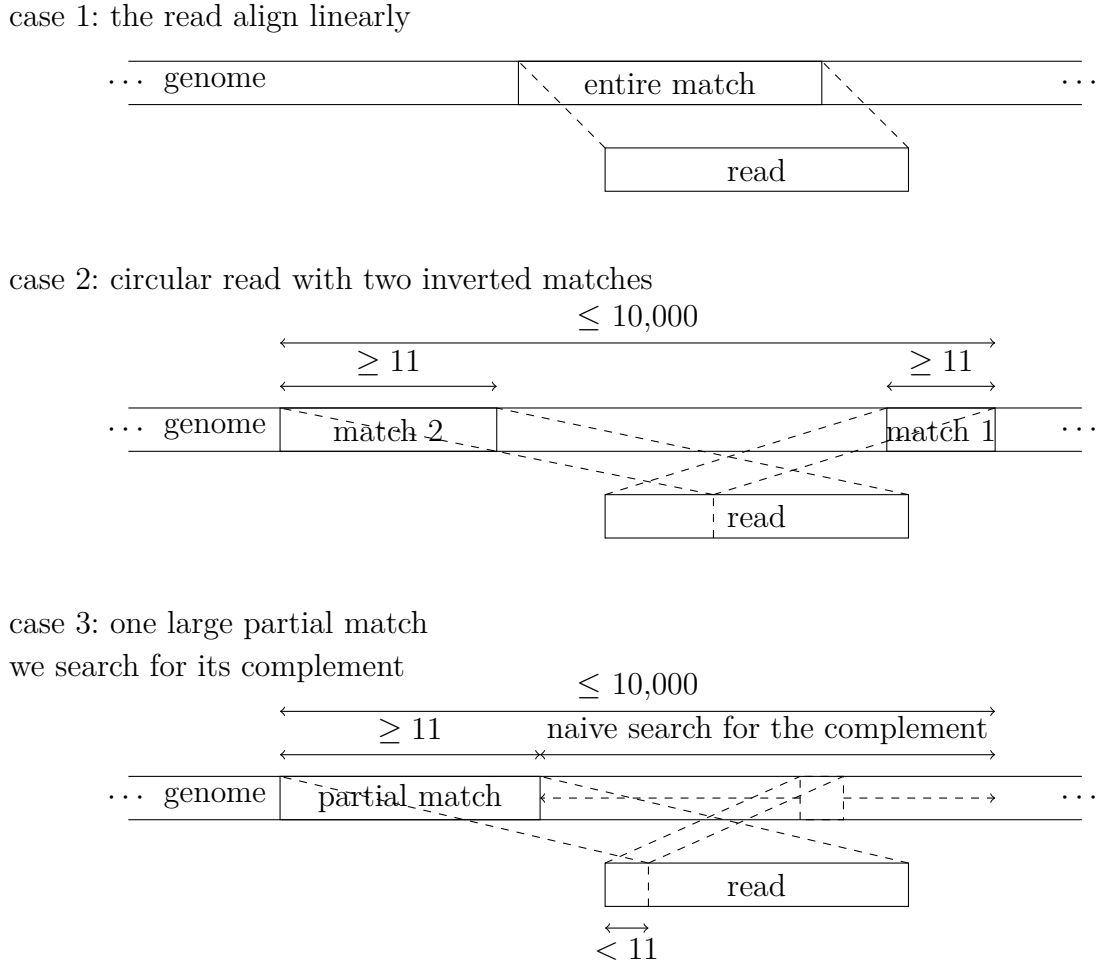


Figure 4.2: Illustration of the three different cases that we distinguish when mapping a read.

This data processing step results into two sequence alignment data files in BAM format that correspond to linear and putative circular reads.

The command line to run the detection of putative circular reads is provided in Code 4.2. There are five arguments to provide in this order: the blastn output in xml format, the two output files in bam format, the reference genome file and the fasta file containing all the reads from the sequencing machine.

```
python blast_analysis.py $OUTPUT".xml" $OUTPUT"_outc.bam" $OUTPUT"
   _outl.bam" $REF $OUTPUT".fasta" $REF_SIZE$
```

Code 4.2: Command line to run the detection of putative circular reads.

4.1.3 First results

The mapping step led to the identification of approximately 80,000 putative circular reads. We note that as it was already stated by Szabo et al. [10], the RNase R treatment is not entirely efficient. Indeed we still obtained a large amount of linear reads in the RNase R treated samples, see Table 4.1. Two RNA-seq experiments were performed with the same protocol A) Pulldown ligase, we decided to merge these results. To count the number of covering reads, we divided by two the sum of reads covering the position in the two samples. Thus, we ignore the positions covered by only one read. For the number of mapped reads we put the average of both samples. We are aware that using the average of two samples is not ideal, but it allowed us to give the same weight at each protocol for the analysis.

Experiments	Circular reads		Linear reads		# of mapped reads
	% of mapped reads	% of genome[1]	% of mapped reads	% of genome[1]	
A) Pulldown ligase	8.3	6.8	91.7	50.3	247,060
B) Pulldown ligase + RNase R treatment	14.7	2.7	85.3	8.7	44,413
C) Total RNA + RNase R treatment	11.4	2.8	88.6	7.3	284,856

[1] The genome size of *P.abyssi* is 1.76Mbp of which at least 79.5% is transcribed. The portion of the genome (% of genome) covered by the mapped reads is indicated for each sample.

Table 4.1: The summary of RNA-seq results and RIP assays using the *Pab1020* antibody

In this table we see that eleven to fifteen percent of reads obtained using a RNAse R enrichment were classified as 'circular' using our computational criteria. These circular reads covered only a minor part of the transcribed genome (2-3%), therefore indicating that the combined experimental and computational criteria are strict. We also analyse the overlap of the putative reads found in each experiment, the results are in Table 4.2. We observe that a large number of RNase R resistant reads interact with *Pab1020*. Indeed 41% of the positions covered by circular reads in total RNA samples treated with ribonuclease R are also covered by circular reads after RIP assays using *Pab1020* antibodies. However the reverse is not true, the pulldown appears to be less specific to circular RNA as only 14 and 10 % of the positions covered by circular reads after RIP assays using *Pab1020* antibodies are covered by RNase R resistant circular reads, respectively from B and C.

These observations imply that almost half of the identified circRNAs interact with *Pab1020* in the cross-linking experiments. Moreover either the RIP assay was not selective enough or the ligase *Pab1020* interacts with some RNAs that are not circRNAs. Both of these hypotheses explain why the genome is much more covered by the reads from experiment A than B and C, see Table 4.1 .

	A	B	C
A	-	35.4%	41%
B	14.2%	-	23.7%
C	9.69%	41.6%	-

Table 4.2: Overlap of the different experiments in percentage. 35.4% of the positions covered by a putative circular reads of B is covered by a putative circular reads of A. A, circular reads after RIP assays using *Pab1020* antibodies; B, circular reads after RIP assay and ribonuclease R treatment; C, circular reads in total RNA samples treated with ribonuclease R.

We are aware that our experimental and data analysis protocols may be prone to unwanted artefacts. Hence, to establish more selective criteria for circRNA identification we merged the sequencing data from all of our experiments to identify the maximum number of circular RNAs.

4.2 Selection of circular junctions

We observed that the circular junctions were frequently shifted by one to three nucleotides between the different reads. This may either reflect the slight heterogeneity in choosing the transcription initiation site, or, alternatively, the presence of an identical base in 5' and 3' termini of a transcript that cannot be solved during the read mapping.

Thus, we grouped together those putative circular reads where the circular junctions were located within three nucleotides. In order to be classified as circular, we only selected junctions that were identified at least in two independent experiments and supported by more than three individual reads that may have different start and end positions. We also requested that at least half of the putative circular reads that aligned entirely inside a given putative junction support the junction.

With these strict and multiple constraints, we identified in *P. abyssi* a total of 133 individual circRNA loci (Figure 4.3) supported by 28,279 circular reads (Figure 4.4).

The command line to run the identification of junctions is provided in Code 4.3. It uses three small codes, the first one takes as input the SAM file of the circular reads, the size of the genome and the genbank file of the genome; it contains its

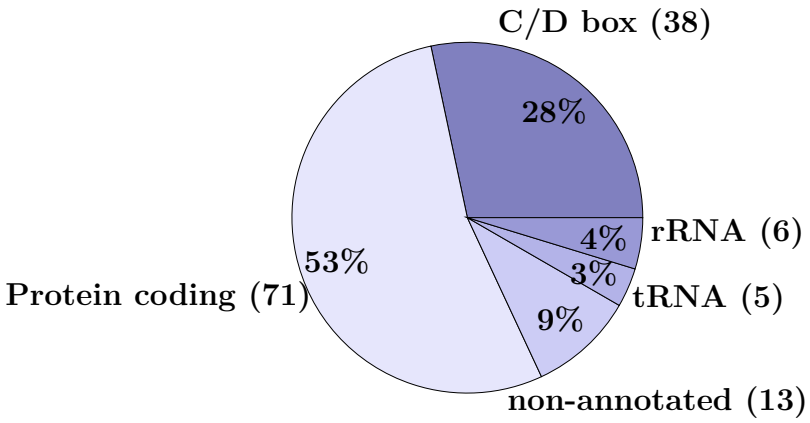


Figure 4.3: Number and percentage of different functional classes of *loci* containing circular junctions identified in our sequencing experiments.

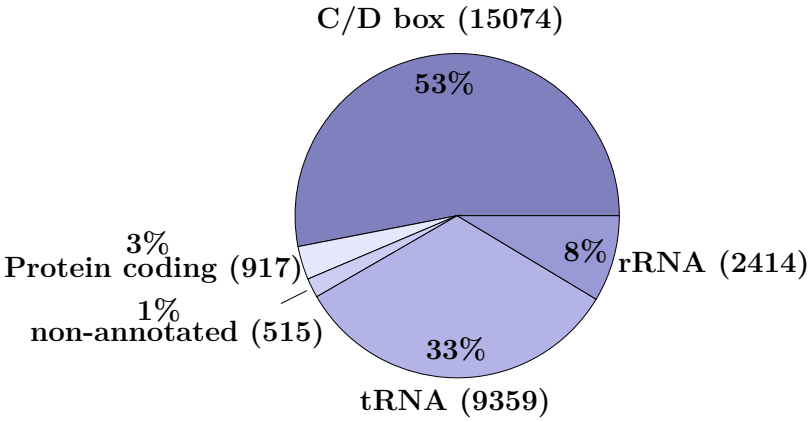


Figure 4.4: Number and percentage of the reads (total 28,279) supporting circular junction of the different functional groups.

annotations. It outputs a file containing for each locus, the junctions present and the number of reads sustaining the junction. Then we use a second code, to select only the junctions corresponding to at least three reads and coming from at least two different samples. Finally, the last code, allows us to apply the last selection criteria; it requires that at least half of the putative circular reads that aligned entirely inside a given putative junction support the junction.

The result file contains the list of the identified junctions with their loci and positions on the genome.

```
samtools view $OUTPUT$"_outc.sorted.bam" > $OUTPUT".tmp"
python stat_circ.py $OUTPUT".tmp" $OUTPUT"_circ.txt" $GBK_REF
   $GBK_NEW $REF_SIZE
```

```
python analyse_circ.py $OUTPUT"_circ.txt" > $OUTPUT"_file.txt"
python analyse_file.py $OUTPUT"_file.txt" $OUTPUT $OUTPUT".out"
```

Code 4.3: Command line to run the identification of circular junctions.

4.3 Identification of functional categories of circular RNAs

4.3.1 Repartition of circRNAs in the different RNAs functional categories

The 133 *P. abyssi* circRNA loci represent five distinct functional groups: C/D Box RNA, non-annotated small RNAs, protein coding RNA, tRNA and rRNA (Figure 4.3). Among these, circular reads were over-represented for 38 circular C/D Box, out of 59 identified by the Lowe Lab in *P.abyssi* genome [66]. From the 46 identified tRNA in *P.abyssi* genome, 5 were covered at least partially by a circular junction: PABt05 (tRNA-His), PABt10 (tRNA-Leu), PABt35 (tRNA-Trp which is known to contain a C/D box intron), PABt39 (tRNA-Leu) and PABt44 (tRNA-Met).

Although 71 loci out of 133 loci correspond to protein coding mRNAs, these were supported only by 3 % of the analyzed reads (Figure 4.4). This approach also led to the discovery of 13 new circular RNAs, marked as non-annotated (NA) in Figures 4.3 and 4.4. The 6 last junctions are non overlapping junctions found on the loci of the 16S rRNA and the 23S rRNA. The constraints applied in our computational pipeline, allow attaining highly selective circRNAs identification.

Numerous circRNAs, including non-coding RNAs, i.e. C/D Box RNA, tRNA-intron and rRNA, were also observed in previous similar RNA-seq study [9]. They studied non treated and RNase R treated RNA-seq samples obtained with a Genome Analyser II (Illumina), from *Sufolobus solfataricus*. Their criteria were to observe two inverted matches that cover the read, and to have at least one circular read supporting the junction from each sample. They identified circRNAs in 37 loci. Among them three are rRNA genes, 5S, 16S and 23S, five are tRNAs, tRNA-Trp, tRNA-Lys, tRNA-Met, tRNA-Pro and tRNA-Ser, those eight junction were from far the most represented junctions they have identified. They also found circRNAs in 12 snoRNA, 11 C/D Box and 1 H/ACA Box and in 7 other non coding RNA. The rest of circRNAs were found in protein coding genes or in intergenic regions, see Figure 4.5.

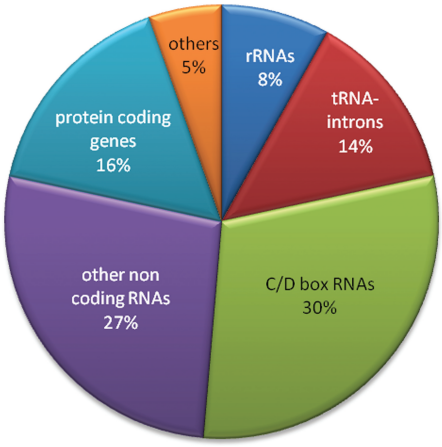


Figure 4.5: Functional characteristics of 37 genes encompassing circular transcripts identified in Danan et al. study [9].

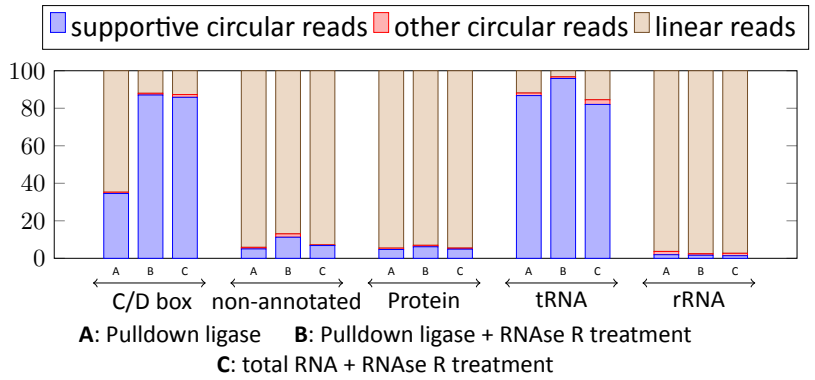


Figure 4.6: Percentage of the reads supporting RNA circular junctions (supportive circular reads) of the different RNA categories. 'Other circular reads' refers to a minority of putative circular reads that fulfil all the computational criteria without supporting the junctions identified. Non-Annotated (NA) refers to previously non-annotated loci.

4.3.2 Enrichment in circular reads with a RNase R treatment

We noticed that circRNAs corresponding to the different functional categories did not behave identically in RNase R-enrichment experiments. Strikingly, the relative portion of circular reads markedly increased after RNase R treatment from 35% to 86% for C/D Box RNAs and was constantly high, around 88% , for tRNAs (Figure 4.6). The fact that RNase R treatment induces an enrichment in the amount of reads supporting circularization junctions in pull-down and total RNA samples further indicates that *Pab1020* RNA ligase specifically associates with circular RNA loci in *P. abyssi* cells.

For the three additional functional groups, this RNase R enrichment for circRNAs was less obvious (Figure 4.6). Note that for the specific case of the tRNA-Trp, the circularization of the encoded-intron occurs simultaneously during the splicing process and linear intron intermediates is not expected to occur. For others RNAs (NA, protein coding and rRNA), the amount of circular reads is too low compared to linear reads for a same locus to allow the enrichment visualization. However, three non-annotated circRNAs (NA7, NA12, NA13 in Table 4.3) out of thirteen showed some enrichment supported by significant amount of reads, see Table 4.3. These three non-annotated circRNAs do not have C/D boxes but they have the same size as C/D Box RNA, between 60 and 70 nucleotides. They are strongly supported, with at least 24 reads supporting the junctions when merging the samples. Thus we are confident that they are not false positives.

4.3.3 Repartition of the circular junctions in the different experiments

The Venn diagram, in Figure 4.7, illustrates the repartition of the 42 enriched (in white) and the 71 non-enriched (in black) identified circular junctions, amongst the different experiments and functional group. In each circle we show which of the 133 identified junctions would have been identified in the experiment analysed separately.

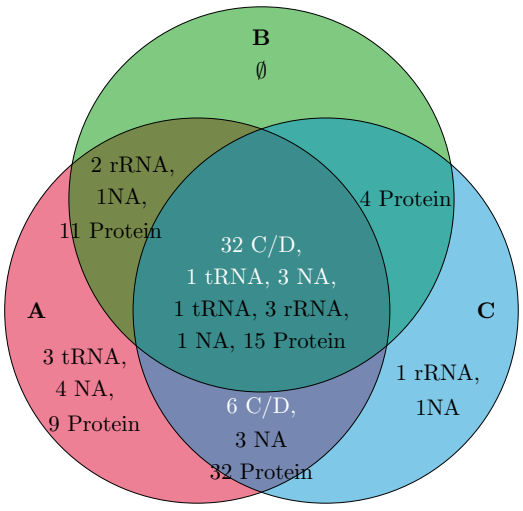


Figure 4.7: Venn diagram summarizing the results of our RNA-seq experiments. A, circular reads after RIP assays using *Pab1020* antibodies; B, circular reads after RIP assay and ribonuclease R treatment; C, circular reads in total RNA samples treated with ribonuclease R. White numbers refer to the categories of the 42 junctions with increased enrichment in RNase R experiments, see Table 4.3.

Even though we require for every junction to be sustained by reads coming from at least two different experiments, this does not imply that they are identified in at least two experiments separately. Indeed, it might occurs that some reads support a junction but do not allow for the identification of the junction by themselves in their experiment. For example if there are many reads in their experiment that support an overlapping junction, then the considered junction might not be identified. Interestingly, 131 of the identified circular loci (98%) were found in a RNA ligase pull-down fraction, suggesting that *Pab1020* is necessary for RNA circularization in *P. abyssi* cells. As expected a vast majority of the enriched identified circular junctions, 36 out of 42, are identified in each experiment taken separately. This enhances our confidence in their identification as circular junctions.

4.3.4 The special case of rRNAs

A high number ($\sim$ 38,000) of circular reads were mapped to ribosomal RNAs (5S, 7S, 16S and 23S rRNAs) but in most cases, localization of the precise position of the junction point from inverted reads was far from evident, possibly reflecting the length and highly structured nature of these RNAs that hinders activity of the reverse transcriptase. However, in the case of the 5S rRNA, we identified 170 inverted reads indicating a specific circularization event between the 5' and 3' extremities of 5S rRNA (with a ten-nucleotide margin). As 5S rRNA interacted with *Pab1020* in cell-free extracts and its circular form has been previously observed [9], this enzyme may participate in 5S rRNA pre-processing via a circular intermediate. This was previously proposed in *A. fulgidus* and *S. solfataricus* for 16S and 23S rRNAs, whose precursors present the BHB motif recognised by the archaeal splicing endonuclease [67].

4.4 Comparisons of our methods with other algorithms

In recent studies [10, 53], different algorithms for identifying circular RNAs have been compared. As detailed in section 3.1.2, they are highly divergent with only 16.8% of identified circular reads from human RNA-seq data are identified by the five algorithms. Hansen et al. [53] finally suggest combining the predictions of circRNA_finder [58] and find_circ [44] to identify circular RNA.

Thus we decided to compare our result with their results on different kind of data. Our method is not adapted to large genomes with splicing such as the Human

Gene	Transcript start	Transcript end	Circle start	Circle end	Circle size	# reads in A[1]	# reads in B[2]	# reads in C[3]	enrichment[4]
sR46	57320 (asRNA)	57597	57374	57441	67	867	292	899	1.05
PABsnRNA44 (sR45)	63444 (ea)	65712	64244	64306	62	15	8	28	7.08
PABsnRNA21 (sR14)	65157 (asRNA)	65282	65218	65278	60	49	16	107	2.87
sR22	63444 (ea)	65712	65333	65397	64	206	17	255	2.41
sR32	67489 (ea)	68026	67951	68012	61	143	25	140	1.97
PABsnRNA3 (sR21)	126284 (ea)	127802	127067	127128	61	9	0	3	15.73
PABsnRNA10 (sR2)	230598 (asRNA)	230710	230632	230696	64	11	2	12	1.94
sR49	235431 (sRNA)	235516	235439	235502	63	35	5	61	1.29
PABsnRNA33 (sR13)	245269 (ea)	246692	245974	246035	61	7	2	8	3.86
sR31	257459 (ea)	258141	258066	258127	61	55	19	100	2.16
PABsnRNA35 (sR29)	318111 (utr5)	318226	318118	318182	64	64	3	35	2.81
PABsnRNA32 (sR4)	473162 (utr5)	473254	473175	473236	61	1	0	3	52.2
PABsnRNA38 (sR58)	539780 (ea)	541847	541770	541831	61	80	14	111	2.15
PABsnRNA40 (sR39)	543825 (sRNA)	543920	543855	543917	62	10	0	2	4.6
PABsnRNA31 (sR20)	553132 (ea)	554479	553656	553721	65	11	1	14	2.67
PABsnRNA12 (sR26)	631448 (sRNA)	631675	631518	631581	63	73	7	181	5.74
PABsnRNA39 (sR60)	631448 (sRNA)	631675	631584	631644	60	218	21	436	7.49
PABsnRNA13 (sR44)	636650 (sRNA)	636775	636702	636762	60	31	5	36	13.58
PABsnRNA17 (sR7)	647252 (ea)	648632	648165	648228	63	4	2	8	43.0
sR25	675393 (asRNA)	675502	675408	675468	60	4	1	5	62.5
PABsnRNA36 (sR55)	910377 (utr5)	910573	910497	910569	72	26	7	27	4.91
PABsnRNA27 (sR35)	949138 (ea)	951058	949199	949261	62	10	2	24	2.83
sR56	960247 (sRNA)	960405	960309	960370	61	30	15	108	5.31
PABsnRNA25 (sR3)	991432 (sRNA)	991508	991446	991505	59	13	1	24	13.74
PABsnRNA1 (sR24)	1024201 (ea)	1028004	1026074	1026133	59	8	10	22	6.93
sR53	1042208 (utr5)	1042365	1042251	1042319	68	2508	720	6159	1.11
PABsnRNA46 (sR38)	1065022 (ea)	1067559	1065728	1065791	63	14	2	9	5.14
PABsnRNA28 (sR37)	1195774 (sRNA)	1195856	1195780	1195842	62	20	1	19	2.32
PABsnRNA23 (sR1)	1209257 (sRNA)	1209332	1209270	1209329	59	21	4	54	36.27
PABsnRNA34 (sR59)	1260087 (utr5)	1260197	1260126	1260195	69	16	1	4	3.21
sR41	1292340 (ea)	1293389	1292356	1292415	59	8	0	4	5.63
PABsnRNA5 (sR11)	1397141 (asRNA)	1397236	1397126	1397188	62	15	1	22	28.62
PABsnRNA29 (sR8)	1403609 (sRNA)	1403742	1403662	1403723	61	3	0	20	113.04
PABsnRNA42 (sR36)	1408146 (ea)	1410562	1409130	1409196	66	20	2	10	9.73
PABsnRNA41 (sR48)	1468599 (ea)	1468802	1468649	1468712	63	26	9	32	1.23
PABsnRNA6 (sR6)	1536318 (asRNA)	1536409	1536388	1536449	61	2	0	2	4.33
PABsnRNA45 (sR34)	1754729 (ea)	1756032	1755871	1755930	59	57	11	112	3.02
PABsnRNA9 (sR12)	1754729 (ea)	1756032	1755929	1755991	62	13	5	15	6.88
PABt35 (sR40)	1330325 (ea)	1330647	1330513	1330584	71	4768	1228	3217	0.89
NA7	621251 (ea)	625587	622461	622526	65	26	4	19	8.39
NA12	1011089 (sRNA)	1011286	1011096	1011158	62	101	19	104	1.91
NA13	1673288 (ea)	1676136	1674520	1674581	61	11	1	12	5.36

[1] Number of circular reads supporting the circularization junction in the pulldown ligase samples.
[2] Number of circular reads supporting the circularization junction in the pulldown ligase + RNAse R treatment sample.
[3] Number of circular reads supporting the circularization junction in the total RNA + RNAse R treatment sample.
[4] Enrichment in reads supporting the circularization junction from the pulldown ligase samples to the total RNA + RNAse R treatment sample. Given by the ratio $\frac{P_C}{P_A}$ with $P_A = \frac{\text{number of circular reads supporting the junction}}{\text{number of reads aligned in the junction}}$ for the pulldown ligase samples and P_C is given by the same formula for the total RNA + RNAse R treatment sample. When the ratio is not defined we wrote 0.

Table 4.3: List of 42 highly significant circular RNA molecules interacting with *Pab1020* RNA ligase in cells.

genome. So we did not try our method on the datasets used by Hansen et al. We used circRNA_finder and find_circ on our datasets, and we also created artificial circular reads to simulate datasets and observe the performance of each algorithms.

4.4.1 Comparison on our datasets

The first thing to note is that their algorithms are much more faster, mainly because of the aligner used, they run in a few minutes while ours run take a few hours. Then we made the comparison sample per sample, so that our comparative analysis of the different experiments does not interfere.

The second thing to observe is that circRNA_finder's output is highly redundant, it does not distinguishes between two highly similar circular junctions. For example a junction that starts at position 65,332 and ends at position 65,397 is considered as

experiment	function	circRNA_finder	find_circ	our pipeline
A1	Total	427 (247 distinct)	24	191
	C/D Box	29	2	39
	tRNA	1	0	7
	rRNA	67	8	12
	non annotated	10	2	23
	others	140	12	110
A2	Total	1319 (751 distinct)	86	161
	C/D Box	31	2	38
	tRNA	3	0	6
	rRNA	121	18	8
	non annotated	16	4	19
	others	580	62	90
B	Total	110 (79 distinct)	7	122
	C/D Box	15	1	27
	tRNA	0	0	4
	rRNA	33	4	7
	non annotated	6	0	7
	others	25	2	77
C	Total	850 (541 distinct)	51	197
	C/D Box	33	1	39
	tRNA	1	0	4
	rRNA	180	38	8
	non annotated	12	4	14
	others	62	8	132

Table 4.4: Number of identified junctions per algorithm and experiment.

a different junction from the one starting at position 65,332 and ending at position 65,398. Thus to make a pertinent comparison we grouped together junctions having the same extremities within a margin of 3 nucleotides at each extremity.

On the contrary, find_circ identifies very few junctions, see Table 4.4. This is due to the fact that find_circ restricts itself to canonical splicing junctions having the motif 'GT-AG', see paragraph 1.2.4.

The results of our pipeline and circRNA_finder look quite similar in Table 4.4. Indeed they identify comparable number of C/B box RNA. However, when looking at the intersection in the prediction our pipeline presents a very small overlap with the two other detection algorithms, Figure 4.8.

The overlap between find_circ and our pipeline is dramatically weak. This is due to the fact that our criteria are very different, they rely mostly on identifying inverted matches presenting the canonical splicing junctions, while we rely only on the matches and on the ratio of circular reads that support the junction.

The overlap between find_circ and circRNA_finder is better, indeed 60 to 80% of the junctions identified by find_circ are identified by circRNA_finder. But their overlap is very thin in number of junctions in comparison to the number of junctions identified by circRNA_finder.

The question that arises is why are our results so different from the other algorithms. Our explanation is that we first identify the same circular reads as circRNA_finder but we reject the junctions, because they do not fit our criteria. Our more stringent criterion is the one requiring that at least half of the circular reads mapping inside a junction support the junction. This criteria reject the areas where circular reads overlap and there is no clear consensus for the junction to identify. In this case we identify a junction only if it is supported by a majority of circular reads, otherwise we do not identify any junctions. Inversely, in these areas, circRNA_finder can identify dozen of circular junctions even thought there might have some ambiguity. Indeed we see in Table 4.4, that circRNA_finder identify up to 180 junctions in rRNAs, while we identify at most 12 junctions in rRNAs.

To confirm our hypothesis we decided to study artificial data, in which we can estimate the risk of obtaining overlapping circular reads.

4.4.2 Comparison on artificial datasets

We generated artificial datasets, first without simulating sequencing errors. We uniformly and at random picked a position in the genome sequence and simulate circular RNA starting at this position. The length was chosen uniformly and at random from 20 to 1,000. Then for each one of this simulated RNA, we simulated 10 copies having different fragmentation cut. When the length was above 100 we considered only a portion of size 100 overlapping the junction. This way we obtained a set of 10,000 simulated circular reads, representing 1,000 different junctions. Then we generated similar datasets by considering different maximal RNA length; 100 and 10,000.

The number of mapped reads and circular junctions identified are summarised in Table 4.5. It is very interesting to observe that the read length has a large impact in the ability of identifying a junction. circRNA_finder is better when the RNAs are not too short (we observe that it misses 95% of the simulated circular RNAs of length less than 50 nucleotides), while our pipeline misses a lot of junctions when the RNAs are too long. Indeed when the reads are too long, above 1,000 in average, they are likely to overlap as the genome is only 1,700,000 nucleotides long and we consider 1,000 junctions. Thus our pipeline identified the circular reads but not the circular junctions.

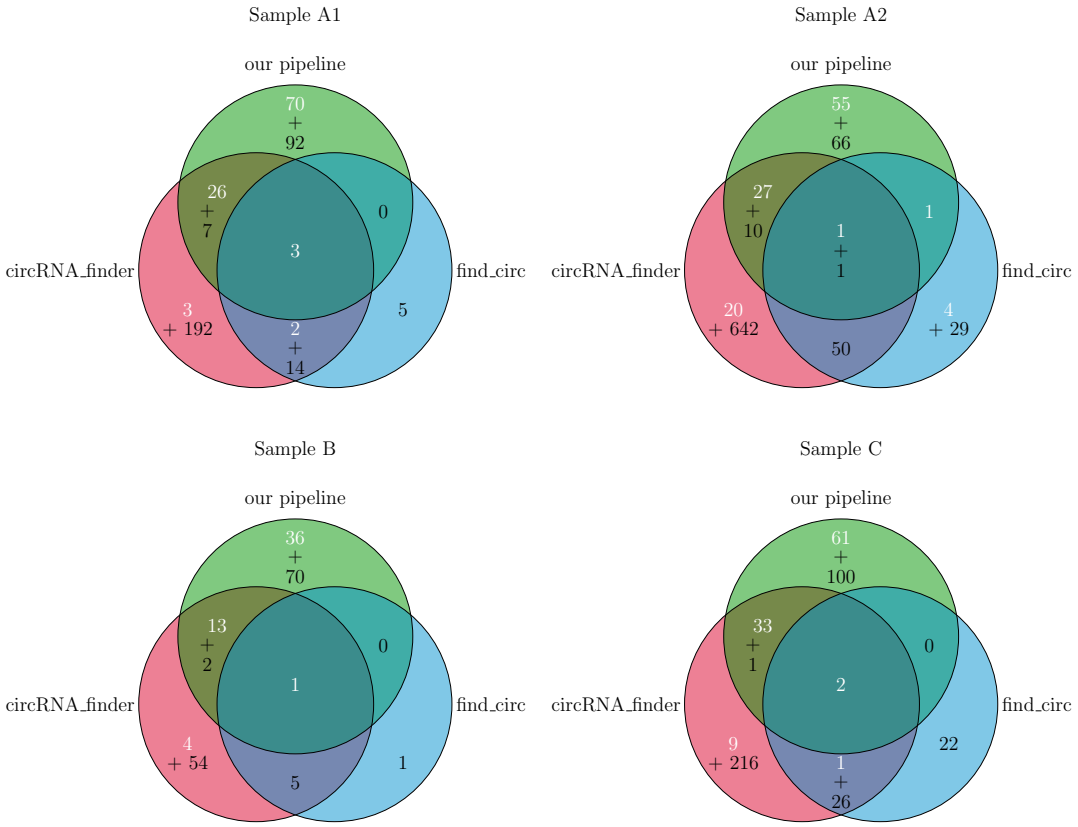


Figure 4.8: Venn diagrams summarizing the results of detection algorithms on four different samples. In white are the number of circular RNA that overlap one of 133 junctions we have identified, in black are the rest of the circular RNA. A, circular reads after RIP assays using *Pab1020* antibodies; B, circular reads after RIP assay and ribonuclease R treatment; C, circular reads in total RNA samples treated with ribonuclease R.

We indeed observe that in all cases our pipeline identified much more circular reads than circRNA_finder. These results on simulated datasets strongly support the absence of false positive in the junctions we have identified. Our pipeline is more accurate in identifying circular reads, we mapped more than 90% of the reads simulated. Then we are more selective in the identification of the circular junctions, thus we have a strong confidence in the junctions we have identified in *P. abyssi*.

We reproduced the same kind of analyses with *wgsim* [68] to simulate reads with errors, we used default parameters; error rate of 2% and mutation rate of 1%. We restrained ourselves to reads having a length up to 1,000, because *wgsim* does not simulate reads of length 10,000. The table 4.6 shows the number of circular reads mapped and the number of junctions identified.

	max read length	100	1,000	10,000
circRNA_finder	number of junctions	605	958	986
	number of circular reads mapped	2,352	5,220	5,258
our pipeline	number of junctions	967	863	508
	number of circular reads mapped	8,649	8,349	8,344
	number of linear reads mapped	1,202	737	692
	unmapped reads	149	914	964

Table 4.5: Comparison of circRNA_finder with our pipeline using error-free simulated circular reads of different size. For circRNA_finder we only know the number of reads that are mapped as circular.

	read length	100	1,000
circRNA_finder	number of junctions	941	937
	number of circular reads mapped	5,123	5,059
our pipeline	number of junctions	867	776
	number of circular reads mapped	7,258	6,531
	number of linear reads mapped	492	607
	unmapped reads	2,250	2,862

Table 4.6: Comparison of circRNA_finder with our pipeline using error simulated circular reads of different size. For circRNA_finder we only know the number of reads that are mapped as circular.

We note that now the read lengths are not random, they are fixed. This is the reason why circRNA_finder achieves much better results for short reads. These reads are all of length 100, so none of them are too short for circRNA_finder to identify them.

Our pipeline is less robust than circRNA_finder to errors. Indeed circRNA_finder identified roughly the same number of circular RNAs. While we identified 10% less circular RNAs and around 20% less circular reads in comparison to the error-free study. However we still identified much more circular reads than circRNA_finder.

Thus a futur work direction to improve our pipeline is to strengthen the robustness to sequencing errors. To do so we could change the parameters of BLAST alignment, but that will also increase the time consumption, or we could try using another aligner.

5 Perspectives

Contents

5.1 Study in other species

5.1.1 Our motivations

Our results on *P.abyssi* provide evidences of the function of *Pab1020* as a ligase that interacts with circRNAs. The protein *Pab1020* is a member of the conserved Rnl3 family of RNA ligases that are predominantly found in hyperthermophiles (archaea, bacteria) and halophiles.

We now aim to prove the *in vivo* implication of a RNA ligase of the Rnl3 family in the formation of circRNAs. To do so we need a genome having a Rnl3 ligase and in which there are genetic tools available to inactivate a gene. *Thermococcus barophilus*, an hyperthermophilic archaea, fill these criteria. The Laboratoire de Microbiologie des Environnements Extrêmes in Brest will inactivate the gene coding for its Rnl3 ligase, with the pop-in/pop-out method [69]. Thus we will study the transcriptomes of *T.barophilus* in the wild type and devoid of Rnl3 protein.

Organism name		Rnl3 protein
Hyperthermophilic achaea	*Thermococcus barophilus*, wild type	yes
Hyperthermophilic achaea	*Thermococcus barophilus*, inactivation of Rnl3	no
Halophilic archaea	*Natrialba magadii*	yes
Halophilic archaea	*Haloferax volcanii*	no
Hyperthermophilic bacteria	*Aquifex aeolicus*	yes
Halophilic bacteria	*Halorhodospira halophila*	yes

Table 5.1: Different organisms for which we plan to do RNA-seq experiments and analysis.

Moreover we decided to study the transcriptome of other organisms, we chose four other organisms, one hyperthermophile and three halophile, expressing or not a Rnl3 protein, see Table 5.1.

Haloferax volcanii presents a particular profile since circular introns have previously been identified during pre-tRNA-Trp processing whereas the Rnl3 family enzyme is absent [70].

The questions we are interested in are:

- Is there a correlation between circRNAs characteristics and the Rnl3 ligases ?
- Do the organisms without Rnl3 ligase have different kind of circRNAs ?

This project has been drawn up in collaboration with two other teams; the Laboratoire de Microbiologie des Environnements Extrêmes in Brest and the Laboratoire de Bioénergétique et Ingénierie des Protéines in Marseille. Indeed the selected species are extremophiles and some of them require special materials to be grown. In Marseille they grew *Aquifex aeolicus* and sent us the cells ready for RNA extraction, we stored it at -20°C. The Laboratoire de Microbiologie des Environnements Extrêmes in Brest is working on the gene inactivation of *Thermococcus barophilus*. We have ordered *Natrialba magadii* and *Halorhodospira halophila* culture pellets at DSMZ, we received them dehydrated and we followed their protocols for cell cultures. The culture of *Natrialba magadii* went flawlessly, but we were not able to grow *Halorhodospira halophila* . This organism is anaerobic, and we may need the help of a specialised laboratory to grow it. As for *Haloferax volcanii*, this organism was already studied in our lab, thanks to Roxane Lestini, the culture protocol is well-established thus we obtained the cells easily.

5.1.2 Cell culture and growth

Natrialba magadii

Natrialba magadii was grown in a medium composed as follow :

- NaCl 200g/L,
- Na_2CO_3 18.5g/L,
- Yeast extract 20g/L,
- Sodium citrate dihydrate 3.4g/L,
- KCl 2g/L,
- $MgSO_4$, $7H_2O$ 1g/L,
- $MnCl_2$, $4H_20$ $3.6.10^{-4}$g/L,
- $FeSO_4$, $7H_2O$ 2.10^{-5}g/L,
- Casamino Acids, 7.5g/L.

After resuspension of the dehydrated pellet, the cells were incubated in the medium at 37°C with an agitation of 150rpm.

To measure the cell growth we measured the Optical Density (OD) at 600nm of the culture medium, using a nanodrop. This measure give us a value that depends on the total cell density, dead or alive. At the beginning of the culture the cells grow slowly, the OD increase is small. Then they reach the exponential growth, they divide actively and the OD increase is important. Finally, they reach the saturation of the medium, they can not grow anymore. The time scale of this scheme depends on the organism, and on the culture conditions. We wanted to collect the cells in the middle of their exponential growth phase. We measured regularly the cell density every 2 hours, in order to obtain a curve as complete as possible we start a culture (culture 1) in the morning and another one (culture 2) 8h later, this way the measures will be complementary. We obtained the growth curve in Figure 5.1, we see that the exponential growth starts between 8 and 16hours and ends around 28h, thus we collected the cells at 22hours of culture.

Halorhodospira halophila

We tried to grow *Halorhodospira halophila* with different pH from 7 to 8.5 and temperatures from 25°C to 37°C. The cultures were permanently illuminated. Moreover we used anaerobic atmosphere generation bags and argon gaz to remove as many oxygen as we could. We were surprised to see that in previous study [71] the authors used a temperature of 42°C while DSMZ advices to use 25°C. We tried to stay as close as possible to that was recommended by DSMZ but none of our culture conditions was a success. We think that this might be because of our anaerobic conditions that were not strict enough. 1L of the medium is composed of:

- KH_2PO_4: 250mg,
- $CaCl_2x2H_2O$: 25mg,
- NH_4Cl: 0.40g,
- $MgCl_2x6H_2O$: 50mg,
- NaCl: 90g,
- $Na_2SO_4x10H_2O$: 10g,
- Na_2CO_3: 3g,
- Na-succinate: 1g,
- Yeast extract 0.25g,
- $NaHCO_3$: 14g
- Solution A: 1mL
- Na_2S: 1g

1L of Solution A is composed of:

- $FeCl_2x4H_2O$: 1.8g,
- $CoCl_2x6H_2O$: 250mg,

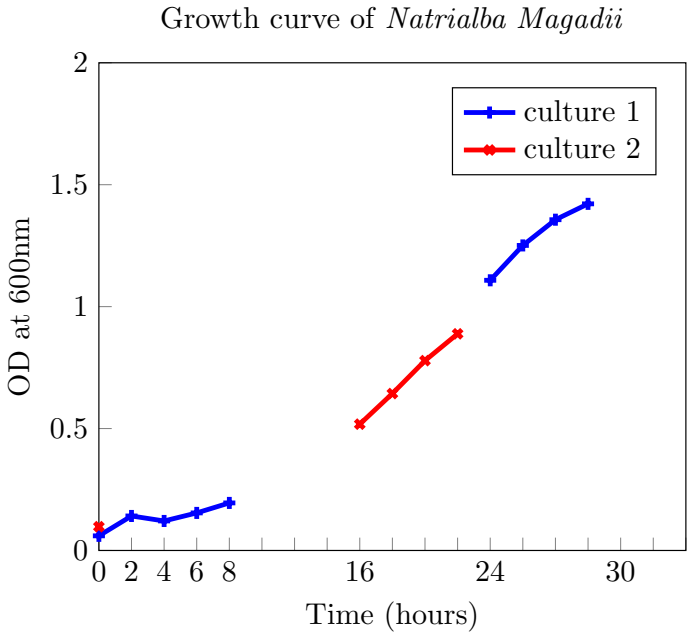


Figure 5.1: Growth curve of *Natrialba magadii*, realised by measuring the OD of the cultures at different time of growth .

- $NiCl_2x6H_2O$: 10mg,
- $CuCl_2x2H_2O$: 10mg,
- $MnCl_2x4H_2O$: 70mg,
- $ZnCl_2$: 100mg,
- H_3BO_3: 500mg,
- $Na_2MoO_4x2H_2O$: 10mg.

Haloferax volcanii

Haloferax volcanii was grown on the YPC medium (yeast peptide complex) at 45°C with 150rpm. 1L of YPC medium is composed of:

- purified H_2O: 300mL,
- 30% salt water solution: 600mL,
- 10X YPC solution: 100mL.

2L of the salt water solution contains:

- NaCl: 480g,
- $MgCl_2$, 6 H_2O: 60g,
- $MgSO_4$, 7 H_2O: 70g,
- KCl: 14g,
- Tris HCl pH 7.5 (1M): 40mL.

204mL of the solution of 10X YPC contains:

- purified H_2O 156mL,
- Yeast extract 10.2g,
- Peptone 2.04g,
- Casamino Acids 2.04g,
- KOH (1M) 3.6mL.

Similarly as for *Natrialba magadii*, we have studied the cell growth in order to collect them during the exponential phase. We obtain the curve showed in Figure 5.2. The exponential growth starts around 10hours of culture and ends around 35hours, thus we collected them at 23hours.

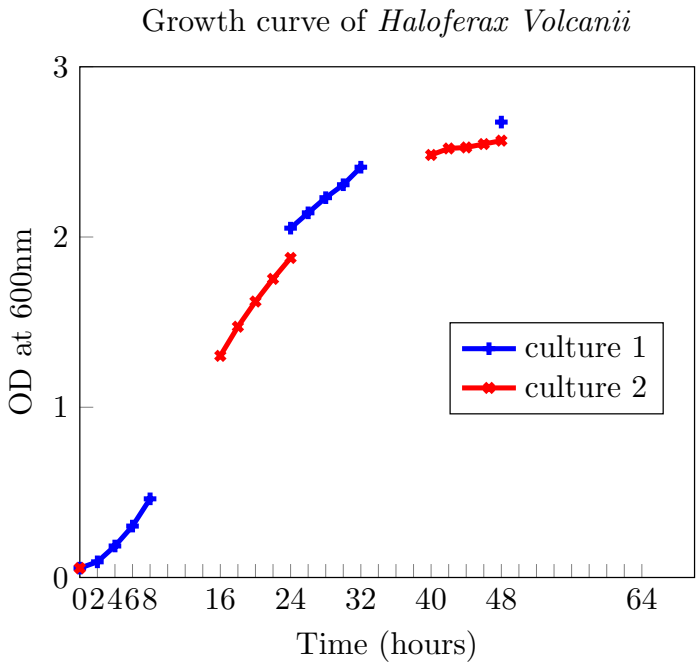


Figure 5.2: Growth curve of *Haloferax volcanii*, realised by measuring the OD of the cultures at different time of growth .

5.1.3 Further perspectives

We do not know yet what kind of circRNAs we will find in these organisms. We expect to obtain numerous circular reads mapping on the rRNAs as we have observed in *P.abyssi*. This was also observed by Danan et al. [9] in *Sufolobus solfataricus*, we note that they also found C/D box circRNAs. The genome of this organism does not present a Rnl3 family protein. However, we found a protein that has 20% of identities with a Rnl3 ligase. Although it is annotated as a DNA-ligase, we suspected it to be a RNA-ligase, at first, *Pab1020* was also wrongly annotated as a DNA-ligase. We present some perspectives on further studies depending on the results obtained in the different organisms:

- For the organism having a Rnl3 family protein, we expect to obtain similar results to what we obtain in *P.abyssi*; numerous small circRNAs of size around 60 to 70 nucleotides. If we indeed identify this kind of circRNAs or other kinds of circRNAs it will be interesting to perform RIP assays to confirm the function of the Rnl3 family protein in the circularization of the RNAs.

- For *Thermococcus barophilus* it will be of particular interest to compare the identified circRNAs in the wild type and in the gene inactivated strain. We

expect to see more circRNAs in the wild type, this would strongly support the implication of the Rnl3 family in the circularization process of circRNA. Otherwise, if we do not observe significant differences in the two RNA-seq analysis, then it might be because the Rnl3 protein is only indirectly implied or because another mechanism is performing the circularization. A complementary analysis would then be to study the gene expression level. This will give us more insight in the impact of the gene inactivation on the metabolism of the organism.

- Our expectations on *Haloferax volcanii* are different as we already know from previous study [8] that there are some circRNAs although it does not have a Rnl3 family protein. Then our aim is to identified these circRNA and compare them with the results of our other studies. Another mechanism, still unknown, is performing the circularization and we are interested in analysing the resulting circRNAs.

5.2 De Bruijn graph, *de novo* analysis

The previous algorithms to identify circRNA all work with the same idea, the mapping of the reads on the genome. Thus, they strongly rely on the genome. However for numerous organisms the reference genome is not available or it has been established but it is not highly reliable, because the organism is not a popular case of study.

A novel approach is to develop a reference-free method to identify circRNAs. The idea is to build the De Bruijn graph of high-quality reads and then to focus on the cycles to detect circRNA. It is important to consider only high-quality reads, because the De Bruijn graph can be misleading when the error rate is important. The De Bruijn graph of parameter k is a graph in which all $(k-1)$-mers present in the reads are nodes and all k-mers present in the reads are edges. We weight each edge by the number of times their corresponding k-mers occur in the reads.

Our first results look promising. We use small datasets, by considering only the reads that maps over the loci of some circRNAs. The Figure 5.3 shows the graph obtained for the reads of the experiment A1, pulldown ligase, that map over the sR46 locus, that we have identified as circular. The correlation between the graph and the genome sequence is explained in the Example 5.2.1. In this example we can easily retrieve the sequence of the circRNA. We did not deeply elaborate the method, so our prototype does not scale well and we were not able to analyse the

whole genome. However, the idea is attractive as it could allow to identify circRNAs without having prior knowledge of the genome of the organism.

This could be interesting for incoming work on identifying circRNAs from RNA-seq.

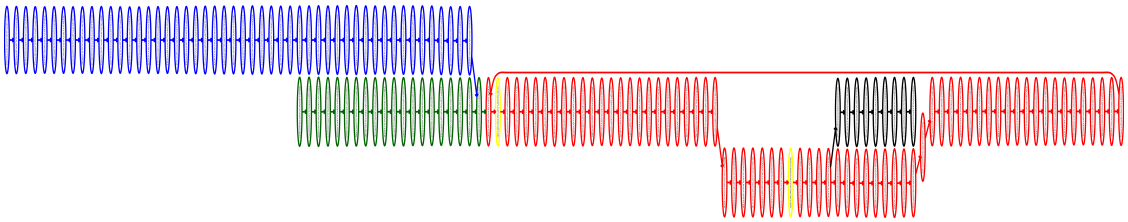


Figure 5.3: De Bruijn graph with parameter k=31, for the reads of the experiment A1 (pulldown ligase), we selected only the reads having a phred quality above 10. We represented the edges supported by at least three 31-mers. In red is the cycle corresponding to the circular RNA, in yellow are the nodes corresponding to the annotated start and end of the RNA. In blue is the part that precedes the circular RNA and in green and black are parts the graph due to an error in some reads.

Example 5.2.1 The RNA sR46 is annotated from 57,375 to 57,438 in *P.abyssi* genome. The sequence of the genome from nucleotide 57,323 to nucleotide 57,440 is:
GTGCTAAAATCAGGTTCTTCTCAAATCAAACATCTTCAAGCTTAAGCTCAGGGCAA
TGAGGAATGAATCCAATGCTGAGCAAAGGCAATGATTGACCCCAGAGTGGCCGAGC
CTCTAT

The part in orange corresponds to the annotated C/D box, it is flanked by red letters that correspond to the circular RNA found with the De Bruijn graph. The part in blue corresponds to the preceding region, it is present in the graph due to linear reads overlapping the junction, there are not any reads overlapping the graph on the right. The parts in green and black of the graph are due to an error in some reads where a T is missing. There is only one edge missing from the black and the green part of the graph to be connected, but the corresponding 31-mer is present only twice in the data thus it is not represented.
green part: CCAGAGTGGCCGAGCCTCTA
black part: GGGCAATGA

Part III

Computation of Minimal Absent Words

6

Words and sequences

Contents

6.1 Definition and notation

Sequences. Let Σ be an *alphabet* - a finite set of symbols, of size σ. An element a of Σ is called a *letter*. Finite sequences of elements of Σ are called *words*, *sequences*, *texts* or *strings*. The *length* or *size* of a word w is the number of its elements (with repetitions), we denote it $|w|$. The i-th element of a word w is denoted by $w[i]$ and i is its *position* in w. We denote by $w[i\,..\,j] = w[i]\,..\,w[j]$ the *factor* of w that starts at position i and ends at position j. If $i > j$, by convention, the word $w[i\,..\,j]$ is the *empty word*, of length 0, denoted by ϵ. A factor v of w is a *proper factor* of w if $0 < |v| < |w|$.

Let w be a word of length n, and u a word of length m, $0 < m \leq n$. We say that there exists an *occurrence* of u in w or that u *occurs* in w, when u is a factor of w. Every occurrence of u can be characterised by a starting position in w. Thus we say that u occurs at the *starting position* i in w when $u = w[i\,..\,i+m-1]$. Conversely, we say that the world u is an *absent word* of w if it does not occur in w.

A *prefix* of w is any word v such that $v = \epsilon$ or v is a factor of w starting at position 0. A *suffix* of w is any word v such that $v = \epsilon$ or v is a factor of w ending at position $n - 1$.

We define the *reverse* of a word $w = w[0]\,..\,w[|w| - 1]$, by $w^r = w[|w| - 1]\,..\,w[0]$, we note that $(w^r)^r = w$.

To concatenate two words, we use the operator '$\cdot$'. For two words $w = w[0]\,..\,w[n-1]$ and $w' = w'[0]\,..\,w'[m-1]$, we define the concatenation of w and w' as $w \cdot w' = w[0]\,..\,w[n-1]w'[0]\,..\,w'[m-1]$.

This operator allows us to define a *power* of a word. The square of a word is the concatenation of two copies of the word, $w^2 = w \cdot w$. More generally, for every positive integer k, a word w to the power of k is denoted w^k .

Graphs. A *graph* $G = (V, E)$ consists of a set $V = \{v_1, \cdots, v_{|V|}\}$ of *nodes* and a set $E \in V^2$ of *edges*, such that (u, v) is an edge from node u to node v, for all $u \neq v \in V^2$. A graph is *directed* if (u, v) is distinct from (v, u). We call (u, v) an *outgoing edge* of node u and an *incoming edge* of node v. For every node $v \in V$, we define the *indegree* of v to be the number of its incoming edges and the *outdegree* of v to be the number of its outgoing edges. We call the *degree* of a node v the sum of its indegree and its outdegree.

A graph is *labeled* when we attach a *label* $\ell(v)$ to each node $v \in V$ and/or to each edge $e \in E$. A label is a set of symbols, usually letters, or an integer. A *path* $P = v_1\,..\,v_{|P|}$ is a sequence of nodes such that $(v_i, v_{i+1}) \in E$ for all $1 \leq i < |P|$. A *cycle* is a path of length at least 3 from a node to itself. A graph that does not contains cycles is called *acyclic*.

A *tree* is an undirected graph $G = (V, E)$ in which any two nodes are connected by exactly one path. In a *rooted* tree, one node is selected as the *root* node, then for each node u the parent of u is the node connected to u in the path from the root to u. Every node, except the root, has exactly one parent. A child of a node u is a node whose parent is u. An *internal* node is a node with at least one child, while a *leaf* is a node without children. Rooted tree may be directed, either making all its edges point away or towards the root; we will use the latter case.

A *trie* is a rooted tree, where every edge is labeled by a character. For any internal node v, the edges leading to its children must have distinct labels.

6.2 Minimal Absent Words

The number of absent words of length at most n is exponential in n. However, the number of certain classes of absent words is only linear in n. This is the case for *minimal absent words* (also called *minimal forbidden words*), that is, absent words in the sequence whose all proper factors occur in the sequence [72]. An upper bound on the number of minimal absent words is known to be $\mathcal{O}(\sigma n)$ [73], where σ is the size of the alphabet. This bound is known to be asymptotically tight [74, 75].

Definition 6.2.1 An absent word u, $|u| \geq 2$, of w is *minimal* if and only if all its proper factors occur in w. We do not consider minimal absent words of length 1 because they correspond to absent letters, we rather use a minimal alphabet such that every letter occurs at least once in w.

MAWs are closely related to repeated pairs.

Definition 6.2.2 *A repeated pair* R in a word w is a triple $< i, j, u >$ such that i and j are distinct starting positions of word u in w. Moreover we have that:

- R is *left maximal* if and only if $w[i-1] \neq w[j-1]$ or $i = 0$ or $j = 0$.
- R is *right maximal* if and only if $w[i+|u|] \neq w[j+|u|]$ or $i+|u| = |w|$ or $j+|u| = |w|$.
- R is *maximal* if and only if it is left maximal and right maximal.

We will now present the relation between minimal absent words and maximal repeated pair.

Lemma 6.2.1 ([76])**.** *Let aub be a minimal absent word of w, with a, b letters and u a word. Then there exist i and j such that (i, j, u) is a maximal repeated pair.*

Proof. If *aub* is a minimal absent word, then by definition *au* and *ub* occur in w; let $i-1$ be the starting position of *au* and j be the starting position of *ub*. We have that u occurs in w at positions i and j. *aub* is absent from w, so the occurrence of u at position i is not followed by b, thus (i, j, u) is a right maximal repeated pair. Similarly, the occurrence of u at position j is not preceded by a. Consequently (i, j, u) is a maximal repeated pair. □

In 2009 Pinho et al. [76] presented an algorithm to compute minimal absent words based on the analysis of maximal repeated pair. Our approach is very similar to theirs; our main contributions have been to achieve an $\mathcal{O}(n)$ time complexity and to provide different algorithms and implementations. The different computation algorithms are presented in chapter 8 and 9.

The set of minimal absent words contains all the information contained in the input sequence. Indeed Mignosi et al [75, 77], have proved that one can retrieve the input sequence from its set of minimal absent words. The algorithm run in time linear to the size of the trie representing the set of minimal absent words, thus in $\mathcal{O}(n)$, with n the size of the input sequence. For more details about this approach and the problem of retrieving a sequence from its set of minimal absent words inspect [78].

6.3 Applications

6.3.1 In Biology

Biologically, absent words may present a spectrum of information. They can be sequences of nucleotides which are hardly tolerated because they negatively influence the stability of the chromatin or other functional genomic conformation; they can represent targets of restriction endonucleases (oligonucleotides which are particularly common in bacterial and viral genomes); or, more generally, they may be short genomic regions whose presence in wide parts of the genome are not tolerated for less known reasons.

Short words of nucleotides may be systematically avoided in large genomic regions for very different reasons. For example, they play important signaling roles that dictate their appearance only in specific positions, such as consensus sequences for the initiation of transcription and replication.

There has been a large number of studies on the biological significance of absent words. One of the first studies suggested that absent words can be used for choosing artificial DNA sequences for molecular barcodes; for species identification and environmental characterisation based on absence; and for identifying potential targets for therapeutic intervention and suicide markers [79]. The most comprehensive study on the significance of absent words is probably [80]; where the authors suggest that the deficit of certain subsets of absent words in vertebrates may be explained by the hypermutability of the genome. Moreover, the analyses in [81] support the hypothesis that minimal absent words are inherited through a common ancestor, in addition to lineage-specific inheritance. In [82], by computing minimal absent words in four human genomes, it was shown that, as expected, intra-species variations

in minimal absent words were lower than inter-species variations. Silva et al. [83] highlighted the existence of three minimal words in the *Ebola* virus genomes which are absent from the human genome. The authors suggest that the identification of such species-specific sequences may prove to be useful for the development of both diagnosis and therapeutics.

Minimal absent words have also been used for phylogenic reconstructions [84]. Crochemore et al. [85] proposed a linear time sequence comparison using minimal absent words. Rahman et al. [86] conducted an experimental study to analyse five different indices using minimal absent words as a similarity measure.

Moreover, in [87], the authors showed that minimal absent words correspond to fine-tuned evolutionary relationships suggesting that they can be more widely used as markers for genomic complexity. Very recently Vergni and Santoni [88] studied the nature of absent words, and introduced the high order absent words, they are absent words whose mutated sequences are still absent words. They showed these words have some peculiar structural features.

6.3.2 In Computer Science

Minimal absent words have been used in different purposes in computer science [89–91]. We will not describe them in details but just give an idea of one of the applications that has been thoroughly studied, the data compression.

This data compression approach was introduced by Crochemore et al. [92] in 2000; they called it the *Data Compression with Antidictionaries (DCA)*. They showed that their compression method runs in practice in linear time, although the worst case scenario is quadratic, and it allows a fast decompression. In 2002, Crochemore et al. [93] improved the compression ratio by considering *almost absent words* as absent and by separately coding their occurrences as exceptions. The DCA method has been developed in several directions, using suffix trees or suffix arrays data structures (see chapter 7 for the definition of these data structures) and achieving different trade-off between compression ratio, memory requirements and decompression time [94–98]. The latter achieves an asymptotic optimality in compression for stationary ergodic sources - whose statistical properties are time independent and can be deduced from a single, sufficiently long, random sample of the process.

6.3.3 Our motivation

Although minimal absent words have been studied for quite a long time, efficient computation algorithms are still missing. Some papers are purely theoretical and do not provide an implementation and others are applied to real data but they do not use efficient implementation. We wanted to close this gap by providing the first implementation of linear time and space algorithms to compute all minimal absent words for an input sequence. The different implementations we propose provide different trade-off of memory consumption and computation time, from a fast parallel algorithm [99] to an implementation in external memory that can run on a desktop computer [100]. They are detailed in chapter 8.

7

Important indexing data structures

Contents

7.1 What are indexing data structures

Indexing data structures are small data structures (usually smaller or around the same size than the input text) used for large texts when one wants to solve many tasks in optimal time. In sequence analysis, indexing data structures are mostly used

for pattern matching. Once the data structure is built, pattern matching queries can be performed in optimal time as many times as wanted.

The first indexes created were based on the factors of the sequence. However a sequence of size n, has an $\mathcal{O}(n^2)$ number of factors. Thus only a small part of the factors can be indexed. Originally, the suffixes have been chosen: indeed there are only $n-1$ suffixes in a sequence of size n, and all the factors can be retrieved as prefixes of a suffix.

A **suffix tree** is a tree in which we store all the suffixes of a sequence. The sequence has a terminator # added at the end such that every suffix ends with #. To each suffix of the sequence corresponds a leaf, such that the suffix is the label of the path from the root to the leaf. The size of such a structure is $\mathcal{O}(n \log n)$, this is too much for large sequences.

In 1973, Weiner [101] introduced the **compact suffix tree**, which is the most popular and the oldest indexing data structure. He showed that it can be constructed in linear time. Few years later McCreight [102] proposed an improvement of the construction algorithm. In 1995 Ukkonen [103] devised a linear-time algorithm for constructing a compact suffix tree. The main steps of the algorithm are still highly similar to those presented by Weiner, but the algorithm is conceptually more intuitive and easy to understand, see 7.2.3 for a short overview of this construction algorithm.

In the early 1990s Gonnet [104] et al. and Manber and Myers [105] proposed another data structure, called the **suffix array**, a space efficient alternative to the compact suffix tree. Shortly after, Burrows and Wheeler introduced the famous **Burrows-Wheeler Transform (BWT)** [106], that is used, amongst others, in the *bzip2* compression algorithm. These two structures are closely related, they are often combined together to form the **FM-index** [107].

7.2 The different variations of suffix trees

7.2.1 Suffix trees

To introduce this data structure, we give a definition adapted from the one Gusfield gave in his well-known book [108]. Usually, in order to distinguish a suffix from any other factor of the sequence, we append at the end of the sequence a terminator (also called sentinel character) #, that is not part of the alphabet. The terminator allows to easily distinguish the suffixes from the others factors of the sequence. Indeed the suffixes are the only factors ending with the terminator. The suffix tree is a tree in which we store all the suffixes of the sequence. This way every leaf corresponds to exactly one suffix of the sequence.

Definition 7.2.1 A suffix tree T of a sequence S of length n fulfills the following properties:

- It has exactly n leaves numbered from 0 to $n-1$;
- Two outgoing edges of a node have edge-labels beginning with different characters;
- The concatenation of the edge-labels on the path from the root to a leaf i is equal to the suffix i $S[i\,..\,n-1]$.

Definition 7.2.2 A suffix tree is a *trie* if every edge label is composed of exactly one character, see an example in Figure 7.1.

A suffix tree is *compact* if every internal node has at least two children. Then edge-labels can contain several characters, see an example in Figure 7.3. We called *implicit* nodes the nodes that are collapsed into the multi-characters edge-labels. The nodes that are present in the data-structure are called *explicit*.

We briefly present some important properties of suffix trees.

Property 7.2.1 For every factor w of S, there is exactly one node $v \in V$ such that w is the label of the path from the root to v. Then v is said to be the corresponding node of w and w is the corresponding factor of v.

Proof. For simplicity we consider a suffix trie. The proof is identical in a compact suffix tree, except that the nodes can be implicit.

Let w be a factor of S, let i be one of its starting position, then by definition there is a leaf, such that the label of the path from the root to this leaf is the suffix $S[i\,..\,n-1]$. We denote v the $|w|^{th}$ node on this path. The label of the path from the root to v is $S[i\,..\,i+|w|-1] = w$.

The unicity of the node v comes directly from the definition of the suffix tree. Two outgoing edges of a node have edge-labels beginning with different characters, thus for each factor w there is only one path starting from the root and labeled w. □

Property 7.2.2 Every internal node having c, $c > 1$, children corresponds to at least $\frac{c(c-1)}{2}$ right maximal repeated pairs (see definition 6.2.2).

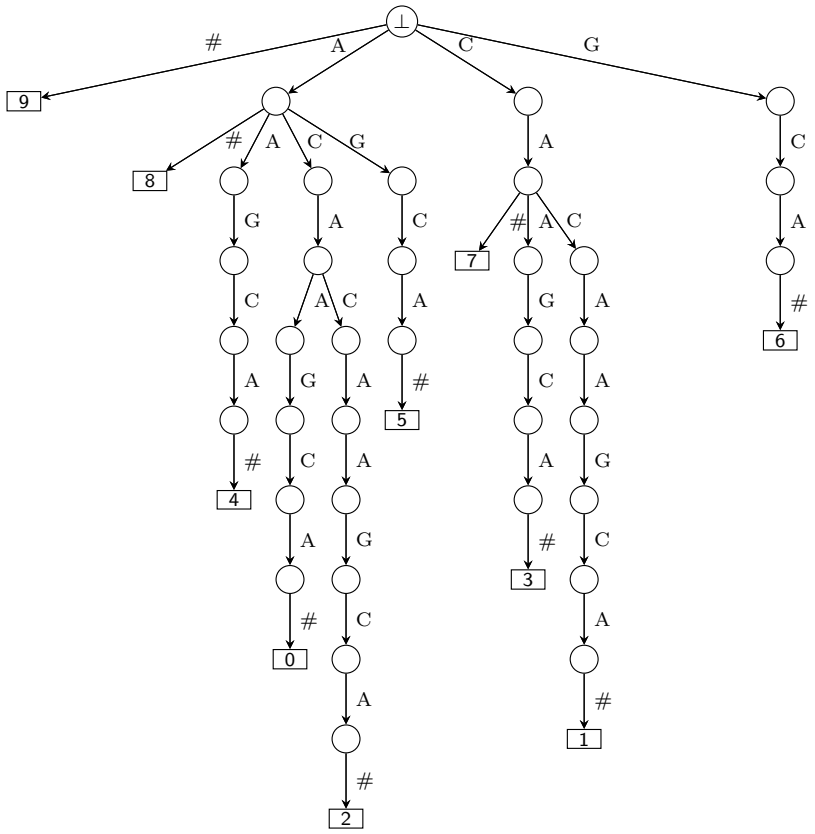


Figure 7.1: The suffix trie for the sequence S=ACACAAGCA#. ⊥ is the root.

Proof. Let v be a node with c children, $c > 1$. We note $v_1, v_2, .., v_c$ its children, and $u_1, u_2, .., u_c$ their corresponding factors (see Property 1). We denote by u the factor corresponding to the node v.

By definition of the suffix tree, u is the longest common prefix of the factors $u_1, u_2, .., u_c$. Let i_1 (resp. $i_2, .., i_c$) be a starting position of the factor u_1 (resp. $u_2, .., u_c$). Then for every couple (i_k, i_ℓ), $1 \leq k \neq \ell \leq c$, $< i_k, i_\ell, u >$ is a right maximal repeated pair. □

The suffix tree is usually augmented with auxiliary edges, called the suffix links. These edges are necessary for an efficient construction of the suffix tree, as we explain in section 7.2.3, and for many applications.

Definition 7.2.3 The *suffix link* of a node v corresponding to a factor αu, with α a letter, goes from the node v to the node corresponding to the factor u. See an illustration in Figure 7.2.

Property 7.2.3 The suffix links of a suffix trie form a trie. We note that this trie is the suffix trie of the reversed sequence.

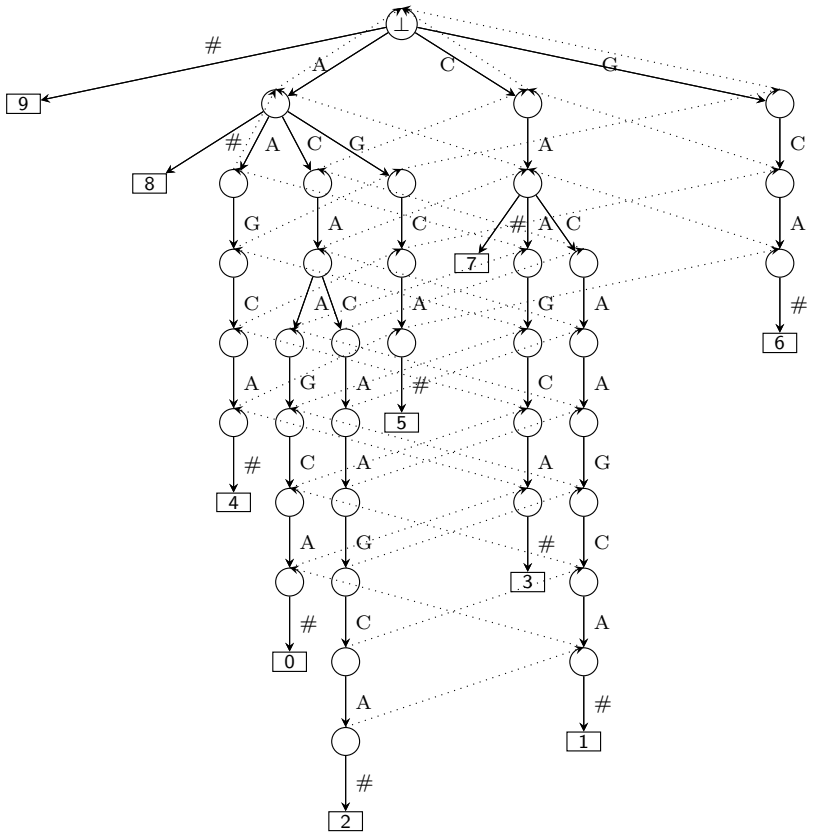


Figure 7.2: The suffix trie for the sequence S=ACACAAGCA# with its suffix links in dashed line.

7.2.2 Compact Suffix Tree

The compact suffix tree has been introduced by Weiner in 1973, it is identical to the suffix trie but all consecutive internal nodes with only one child are collapsed into one, thus the edge labels are strings. An internal node in a suffix trie, can either become an internal node in the compact suffix tree with at least two children, then this node is called *explicit*. Or if it has only one child, it will collapse into an edge with a multi-characters label, then in the compact suffix tree it is called an *implicit* node.

The compact suffix tree, for now on we will refer to it as the suffix tree, occupies an amount of space linear in the size of the input. Indeed, for a sequence of size n there are exactly n leaves, one per suffix. Then in a compact suffix tree, every internal node has at least two children, thus there are at most n explicit internal nodes. Consequently, the whole structure has a linear number of nodes. However the edge-labels are not of linear size, thus to achieve a linear space complexity, there are not stored as sequences, instead they are stored as a pair of positions corresponding to the starting position and the ending position of the edge-label in the input sequence. The compact suffix tree can be build in linear time and

space, the first algorithm was presented by Weiner [101] and has been improved by McCreight [102], Ukkonen [103] and others afterwards. In the next section we briefly explain the Ukkonen construction algorithm, because it is the more intuitive and straightforward, we will use the this algorithm later in section 9.5.

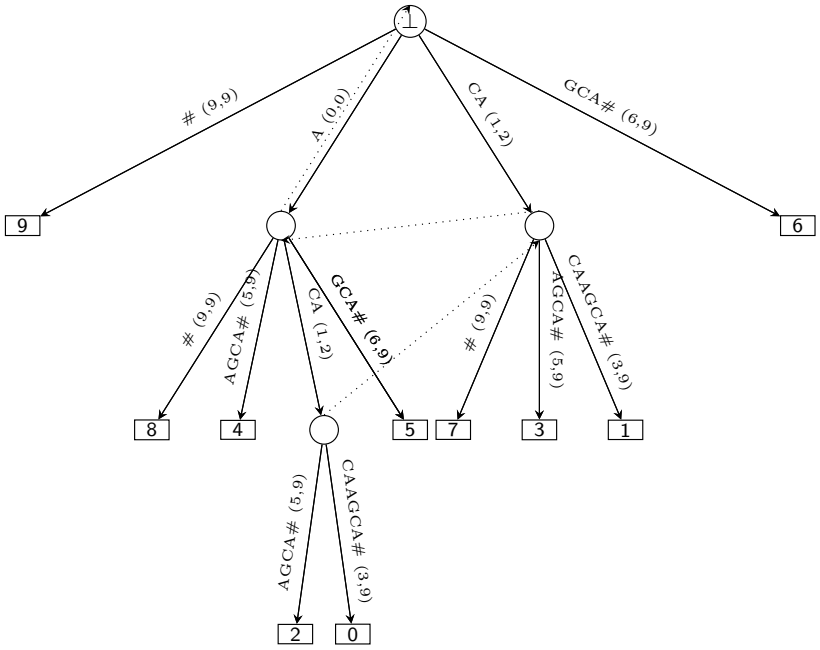


Figure 7.3: The compact suffix tree for the sequence S=ACACAAGCA#.

7.2.3 Ukkonen construction algorithm

We briefly present Ukkonen construction algorithm of the suffix tree. We present the main ideas, the proofs can be found in the original paper [103].

Ukkonen's construction algorithm works on-line, it constructs the suffix tree from left to right and incrementally constructs the suffix trees for the prefixes of the input sequence S seen so far.

An important point raised by Ukkonen is that once a leaf is created, it will remain forever a leaf after all future right extensions. This implies that every time a letter is added at the end of the text the edges leading to existing leaf must be extended by that symbol. To handle this, Ukkonen introduced the *open edges*, whose labels ends at the current and continuously growing end of the text.

To update the tree, it maintains a node, called the *active point* which is the node corresponding to the longest repeated suffix in the sequence (it can be explicit or implicit).

An iteration of Ukkonen's algorithm works on all suffixes between the active point and up to the node corresponding to the shortest suffix that already occurs followed by the added letter, the *end point*. It travels by following the suffix links while performing the following tasks:

- If the active point is implicit, then it makes it explicit. Moreover if the last but one node created has no suffix link, then it adds a suffix link from the last but one node created to the last node created.
- It creates a new leaf and a new edge from the active point to the new leaf, with label starting with the new letter. It adds a suffix link from the last but one leaf created to the last leaf created.
- Then it moves the active node sideways by following the suffix link.

Once the end point is reached, the active node is moved down the edge whose label starts with the new letter, the end point is now the new active point.

We recall that to fulfil the linear space requirement, the edge labels are represented by pairs of offsets of first and last symbol of the label in the input string, except for the edges leading to leaf, they have only the offset corresponding to the first position.

Figures 7.4 to 7.6 illustrate the construction of the suffix tree for the sequence $\overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{\text{S=ACACAAGCA\#}}$.

Ukkonen showed that its construction algorithm is linear in the size of the input text.

7.2.4 Applications

The suffix tree is used for pattern matching, it allows to find a pattern P of size m in time $\mathcal{O}(m + \#occ)$, with $\#occ$ the number of occurrences of the pattern. To do so it travels the tree from the root following the path corresponding to the pattern P. If there is not such path in the tree, then the pattern does not occur in the sequence. If the path leads to a leaf, then the pattern is unique, it occurs only once in the sequence. If the path leads to an internal node or inside an edge, then the pattern occurs as many times as there are leaves in the subtree. The compact suffix tree has many other applications for text algorithms (see [108]).

In spite of its linear space consumption, the compact suffix tree, uses too much space to be applied to huge sequences. Kurtz implemented in 1999 [109] a suffix tree that, for a sequence of length n, uses $10n$ space in average and $20n$ space in the worst case. This was a major improvement, but it stays too expensive for a sequence like the whole Human Genome (3 Go), it requires around 45Go of memory.

7.3 Suffix arrays

7.3.1 Presentation

The suffix arrays were introduced by Gonnet et al. [104] and Manber and Myers [105] in 1993 as a space efficient alternative to suffix trees. Let S be a sequence of size n over a finite and ordered alphabet Σ of size σ. The Suffix Array (SA) of S is an integer array of size n storing the starting positions of all lexicographically sorted suffixes of S. For all $1 \leq r < n$, we have $S[\mathsf{SA}[r-1]\,..\,n-1] < S[\mathsf{SA}[r]\,..\,n-1]$. This array occupies $n \log n$ bits, thus if $n < 2^{32}$ it can be stored in $4n$ bytes, much less than the suffix tree. We note that this array corresponds to the array made by the leaves of a the compact suffix tree, in the order they are seen during a depth first traversal.

Example 7.3.1 The suffix array for the sequence $S = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{\mathrm{ACACAAGCA\#}}$.

i	SA[i]	suffixes
0	9	#
1	8	A#
2	4	AAGCA#
3	2	ACAAGCA#
4	0	ACACAAGCA#
5	5	AGCA#
6	7	CA#
7	3	CAAGCA#
8	1	CACAAGCA#
9	6	GCA#

The suffix array alone can be used for pattern matching. Given a pattern P of size m the search can be done by dichotomy. The search is completed in at most $\log n$ steps, each containing at most m comparisons. Thus the time complexity of the pattern matching is $\mathcal{O}(m \log n)$. Then to output the starting position of each occurrence it take $\mathcal{O}(1)$ time.

The suffix array is a bijection that associates a position in the sequence to the rank of the suffix starting at this position. Thus, it exists an inverse bijection, that associates the rank of a suffix to its starting position in the sequence. The inverse bijection is stored as an array denoted by the Inversed Suffix Array (iSA), we have $\mathsf{iSA}[\mathsf{SA}[i]] = i$ for all $0 \leq i < n$.

7.3.2 Different construction algorithms

As mentioned above the suffix array can be obtained directly from the suffix tree, thus it can be constructed in linear time and space. However, as the suffix tree requires a lot of space, non-linear algorithms are faster than this naive approach. The first construction algorithms of the suffix array were in time $\mathcal{O}(n \log n)$. The first linear time construction algorithms have appeared simultaneously in 2003: Kärkkäinen and Sanders [110], Kim et al. [111] and Ko and Aluru [112]. Construction algorithms of suffix arrays have been thoroughly reviewed by Puglisi and al. in 2007 [113], we briefly present the three main categories:

- Prefix doubling: the suffixes are first ordered according to their prefixes of size 2. Then, knowing this result, they are ordered according to their prefixes of size 4, then 8 etc... The process is repeated until we obtain a total order. They are at most $\log n$ steps to sort all the suffixes, thus the time complexity is $\mathcal{O}(n \log n)$. The construction algorithm initially proposed by Manber and Myers [105] is based on this idea.

- Recursive algorithms: they were the first to achieve linear worst time computation. The idea is to construct two subsequences U and V from S, such that once we have constructed the suffix array of U we can deduce those of V and finally S. Since U is chosen such that $|U| < S$, the overall time requirement is $\mathcal{O}(n)$. For example Kärkkäinen [110] choose to sort, using a bucket sort of depth 3, the suffixes starting at positions congruent to 1 or 2 modulo 3. Once these suffixes are sorted they deduce the order of the suffixes starting at positions congruent to 0 modulo 3, and they merge all the suffixes to find the suffix array.

- Induced copying: the key insight is similar to the recursive algorithms i.e. the complete sort of a selected subset of suffixes is used to induce a complete sort of the whole set of suffixes. The main difference is that their approaches here are non recursive, an example is Manzini and Ferragina in 2004 [114]. In general, these methods are very efficient in practice, but they may have worst-case asymptotic complexity of $\mathcal{O}(n^2 \log n)$.

One of the fastest algorithm for the construction of the suffix array, is SA-IS of Nong and al. [115]. It is a recursive algorithm based on a method of induced sorting. The careful implementation, by Yuta Mori [116], outperforms the other construction approaches available nowadays.

Dynamic algorithm

In 2010, Salson et al. [117] proposed an algorithm to construct the suffix array in a dynamic way, i.e. by allowing updates in the input sequence and editing the suffix array accordingly without rebuilding everything from scratch. Their algorithm has a poly-logarithmic time complexity for the updates, and thus it represents an efficient way to obtain the suffix array of a sequence knowing the suffix array of a very similar sequence (by the edit distance). However, the time complexity is not satisfying for a streaming algorithm, as it will not permit to achieve a linear time amortised complexity. When the number of update operations is too big, it is faster to rebuilt the suffix array from scratch. This is why in chapter 9, we choose to use a suffix tree and not a suffix array to perform pattern matching in a sliding window.

External Memory algorithm

External Memory model We use a model of computation detailed by Vitter in [118]. By M we denote the internal memory or RAM size and by B the external memory (disk) block size, both measured in units of $\Theta(\log n)$-bit words. We further assume that $M = \Omega(\log n)$ and $M = \mathcal{O}(n)$. In the external memory model, each transfer of B words between memory and disk is called an Input Output Operation (IO). Hence, an algorithm's complexity is mainly measured in IOs.

Algorithms External memory algorithms can become very useful when the input sequence is huge and the internal memory is of limited size. Bingmann et al. [119] proposed an adaptation of the algorithm SAIS in external memory, creating the eSAIS algorithm. For a sequence of size n it can compute the suffix array in time $\mathcal{O}(n \log_{\frac{M}{B}} \frac{M}{B})$ and with $\mathcal{O}(\frac{M}{B} \log_{\frac{M}{B}} \frac{M}{B})$ IOs. eSAIS is IO-optimal but it uses large amount of disk, thus Kärkkäinen et al. [120] have recently proposed a new algorithm that is much faster and with less disk usage. They achieve this by using parallel implementation, thus their implementation is much faster, even though it is not IO-optimal.

7.4 Longest Common Prefix (LCP) arrays

7.4.1 Presentation

Pattern matching with the suffix arrays is space efficient but not as fast as with the suffix tree. An additional array is needed to achieve the search of a pattern P of size m in a suffix array in time $\mathcal{O}(m + \#occ)$. This additional structure is called the Longest Common Prefix (LCP) array.

Definition 7.4.1 The longest common prefix function is such that $\text{lcp}(r, s)$ denotes the length of the longest common prefix between $S[\mathsf{SA}[r]\,..\,n-1]$ and $S[\mathsf{SA}[s]\,..\,n-1]$, for all $0 \leq r \leq s < n$ and 0 otherwise.

We note that $\text{lcp}(r, s)$ corresponds to the depth of the longest common ancestor of the nodes corresponding to the suffixes starting at r and s.

The LCP array of S is defined by $\mathsf{LCP}[r] = \text{lcp}(r-1, r)$ for all $1 \leq r < n$, and $\mathsf{LCP}[0] = 0$.

Example 7.4.1 The suffix array and the LCP array for the sequence $S = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{\text{ACACAAGCA\#}}$.

i	$\mathsf{SA}[i]$	$\mathsf{LCP}[i]$	suffixes
0	9	0	#
1	8	0	A#
2	4	1	AAGCA#
3	2	1	ACAAGCA#
4	0	3	ACACAAGCA#
5	5	1	AGCA#
6	7	0	CA#
7	3	2	CAAGCA#
8	1	2	CACAAGCA#
9	6	0	GCA#

Remark 7.4.1 We consider k, r, s, with $k \geq 0$ and $0 \leq r \leq s < n$, if $\text{lcp}(r, s) = k$ then there exist i in $(r, s]$ such that $\mathsf{LCP}[i] = k$.

It has been proved by Abouelhoda [121] that every algorithm that uses a suffix tree as data structure can systematically be replaced with an algorithm that uses an *enhanced suffix array* and solves the same problem in the same time complexity. The *enhanced suffix array* is the suffix array plus the LCP array and the child table. We now introduce the child table.

Definition 7.4.2 An interval $[i, j]$, $0 \leq i \leq j < n$, is an *lcp-interval* of lcp value ℓ if:

- $\mathsf{LCP}[i] < \ell$,
- $\mathsf{LCP}[k] \geq \ell$ for all $i < k \leq j$, with equality for at least one k if $i \neq j$,
- $\mathsf{LCP}[j+1] < \ell$

We notice that lcp-intervals correspond to subtrees in the suffix tree, thus lcp-intervals can be embedded. Let $[i, j]$ be an lcp-interval of lcp value ℓ. Each index k, such that $i + 1 \leq k \leq j$ with $\mathsf{LCP}[k] = \ell$ is called an *ℓ-index*. An lcp-interval $[g, d]$ of lcp value m is *embedded* in $[i, j]$ if $i \leq g < d \leq j$ and $m > \ell$. Then $[i, j]$ is called the *enclosing* interval of the interval $[g, d]$. If there is no other intervals embedded in between $[i, j]$ and $[g, d]$, then $[g, d]$ is a *child interval* of $[i, j]$.

Lemma 7.4.1. *Let $[i, j]$ be a ℓ-interval. If $i_1 < i_2 < \cdots < i_k$ are the ℓ-indices of $[i, j]$ then the child intervals of $[i, j]$ are $[i, i_1 - 1], [i_1, i_2 - 1], \cdots, [i_k, j]$.*

The child table, child, is an array of triplets (*up*,*down*,*next*) such that for all $0 \leq i < n$:

- $\mathsf{child}[i].up = \min\{q \in [0, i-1] | \forall k \in [q+1, i-1], \mathsf{LCP}[k] \geq \mathsf{LCP}[q] > \mathsf{LCP}[i]\}$
- $\mathsf{child}[i].down = \min\{q \in [i+1, n] | \forall k \in [i+1, q-1], \mathsf{LCP}[k] > \mathsf{LCP}[q] > \mathsf{LCP}[i]\}$
- $\mathsf{child}[i].next = \min\{q \in [i+1, n] | \forall k \in [i+1, q-1], \mathsf{LCP}[k] > \mathsf{LCP}[q] = \mathsf{LCP}[i]\}$

Let $[i, j]$ be an lcp-interval of lcp value ℓ with ℓ-indices $i_1 < i_2 < \cdots < i_k$. The meanings of these formal definitions are as follows.

- $\mathsf{child}[\ell].up$ stores the first index of the second child interval of the longest lcp-interval ending at index $\ell - 1$. Thus $\mathsf{child}[j + 1].up = i_1$.
- $\mathsf{child}[\ell].down$ stores the first index of the second child interval of the longest lcp-interval starting at index ℓ. Thus $\mathsf{child}[i].up = i_1$.
- $\mathsf{child}[\ell].next$ stores the first index of the next sibling interval of the longest lcp-interval starting at index ℓ if and only if the interval is neither the first child nor the last child of its parent. Thus for all q, $1 \leq q < k$, $\mathsf{child}[i_q].next = i_{q+1}$.

The child table is necessary to simulate all kinds of suffix tree traversals very efficiently (more details can be found in Abouelhoda et al. paper [121]).

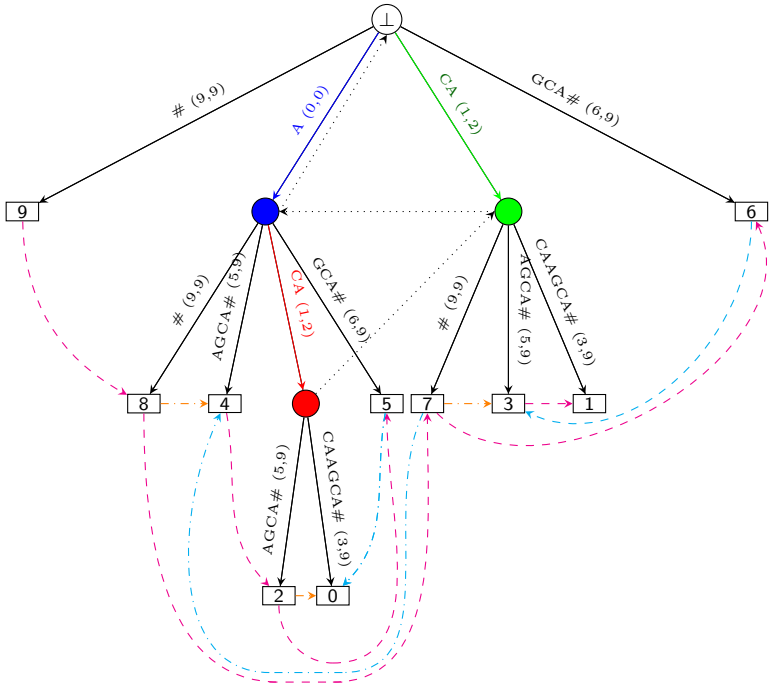
Example 7.4.2 Consider the enhanced suffix array for the sequence $S = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{\mathrm{ACACAAGCA\#}}$. There are:

- one lcp-interval of lcp-value 1 is [1,5], it is the subtree with the blue root. It has three 1-indices: 2, 3 and 5,
- one lcp-interval of lcp-value 2 is [6,8], it is the subtree with the green root. It has two 2-indices: 7 and 8,

- one lcp-interval of lcp-value 3 is [3,4], it is the subtree with the red root. It has one 3-index:4 .

The rest are lcp-interval of size 1, there is one such interval for each suffix (or leaf in the tree).

i	SA$[i]$	LCP$[i]$	child	suffixes
0	9	0	($\emptyset$, $\emptyset$, 1)	#
1	8	0	($\emptyset$, 2, 6)	A#
2	4	1	($\emptyset$, $\emptyset$, 3)	AAGCA#
3	2	1	($\emptyset$, 4, 5)	ACAAGCA#
4	0	3	($\emptyset$, $\emptyset$, $\emptyset$)	ACACAAGCA#
5	5	1	(4, $\emptyset$, $\emptyset$)	AGCA#
6	7	0	(2, 7, 9)	CA#
7	3	2	($\emptyset$, $\emptyset$, 8)	CAAGCA#
8	1	2	($\emptyset$, $\emptyset$, $\emptyset$)	CACAAGCA#
9	6	0	(7, $\emptyset$, $\emptyset$)	GCA#



For an index i, we denote by v_i the leaf corresponding to the suffix starting at position i. When defined, the arrow (v_i, child$[i].up$) is in cyan, the arrow (v_i, child$[i].down$) is in orange and the arrow (v_i, child$[i].next$) is in magenta.

7.4.2 Construction algorithms

As we have seen in the previous section, there are plenty of algorithms to construct suffix arrays. The LCP array can be build at the same time as the suffix array;

that is the case in the augmented eSAIS algorithm [119]. Other algorithms use the suffix array as an input and build the corresponding LCP array. The faster algorithm to compute the LCP array from the suffix array in external memory is LCPscan [122]. Its complexity is quadratic in theory; $\mathcal{O}(\frac{n^2}{M \log_\sigma n} + n \log_{\frac{M}{B}} \frac{n}{B})$ in time and $\mathcal{O}(\frac{n^2}{MB(\log_\sigma n)^2} + \frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B})$ in IOs, and only $16n$ bytes of disk usage. However in practice, the quadratic part does not dominate the computation time when the size of the input is less than 100 times the size of the RAM.

7.5 Burrows-Wheeler transform (BWT)

7.5.1 Presentation

The Burrows-Wheeler transform (BWT) was introduced in 1994 [106]. It is used in everyday data compression techniques like bzip2. The BWT reorganises the input to obtain a text that is much easier to compress. To obtain the BWT of a sequence S, we consider all its circular permutations and we sort them lexicographically: therefore they form a matrix of letters. The last column of this matrix is the BWT, the first column is called F. We observe that there is a strong relationship between the BWT and the suffix array. Indeed for all $0 \leq i < n, \mathsf{BWT}[i] = S[\mathsf{SA}[i] - 1]$, if $\mathsf{SA}[i] > 0$ and '#' otherwise.

The BWT is usually easier to compress than the input itself, because it tends to gather identical letters. For example in English the letter 'h' is often preceded by the letter 't', thus in a text written in English, the suffixes starting with an 'h' are likely to be preceded by a 't'. Consequently we are likely to obtain consecutive 't's in the BWT in the interval corresponding to the suffixes starting with an 'h'.

By using a three-step compression algorithm, the BWT can be compressed in space upper bounded by $nH_k(S)+o(n)$, with $H_k(S)$ the k-th order empirical entropy of the sequence S. The value $nH_k(S)$ represents a lower bound to the compression one can achieve using codes which depends on the k most recently seen symbols.

Example 7.5.1 The suffix array, the F column and the BWT for the sequence $S = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{\mathrm{ACACAAGCA\#}}$.

i	SA[i]	F[i]	permutations	BWT[i]
0	9	#	#ACACAAGCA	A
1	8	A	A#ACACAAGC	C
2	4	A	AAGCA#ACAC	C
3	2	A	ACAAGCA#AC	C
4	0	A	ACACAAGCA#	#
5	5	A	AGCA#ACACA	A
6	7	C	CA#ACACAAG	G
7	3	C	CAAGCA#ACA	A
8	1	C	CACAAGCA#A	A
9	6	G	GCA#ACACAA	A

The transformation is reversible. The input sequence can be retrieved from the BWT. An essential property of the BWT is called the LF-mapping property. It says that for each letter, the order in the F column is the same as in the BWT. This property allows the reversibility of the BWT.

Property 7.5.1 LF-mapping: The k^{th} letter 'c' in F and the k^{th} letter 'c' in BWT correspond to the same occurrence of the letter 'c' in the input sequence. More formally for an index i, $0 \leq i < n$, we denote by 'c' the starting letter of its corresponding suffix ($S[\mathsf{SA}[i]] = c$) and k the rank of this letter among the other 'c' in the column F. Then the k^{th} letter 'c' in the BWT is at position $\mathsf{iSA}[\mathsf{SA}[i]+1]$ if $\mathsf{SA}[i] < n-1$ and $\mathsf{iSA}[0]$ otherwise.

Proof. Let us consider a letter that occurs at least twice, let i and i' be two of its occurring positions. With no loss of generality, we suppose that $S[i]$ occurs before $S[i']$ in F. We denote by s, respectively s', the suffix of S starting at position i, respectively, i'. We denote $\ell = lcp(i, i') \geq 1$ and $w = S[i+1\,..\,i+\ell-1]$, possibly empty. We can decompose the suffixes s and s' into $s = S[i].w.S[i+\ell\,..\,n-1]$ and $s' = S[i'].w.S[i'+\ell\,..\,n-1]$. We know that $s < s'$ thus $S[i+\ell] < S[i'+\ell]$, therefore the suffix starting at position $i+\ell$ is lexicographically smaller than the one starting at position $i'+\ell$. Consequently $S[i]$ occurs before $S[i']$ in BWT. □

Example 7.5.2 The arrays SA, iSA, F and BWT for the sequence
$S = \overset{0}{\mathrm{A}_0}\overset{1}{\mathrm{C}_0}\overset{2}{\mathrm{A}_1}\overset{3}{\mathrm{C}_1}\overset{4}{\mathrm{A}_2}\overset{5}{\mathrm{A}_3}\overset{6}{\mathrm{G}_0}\overset{7}{\mathrm{C}_2}\overset{8}{\mathrm{A}_4}\overset{9}{\#}$.

i	SA$[i]$	iSA$[i]$	permutations	F$[i]$	BWT$[i]$
0	9	4	$\#A_0C_0A_1C_1A_2A_3G_0C_2A_4$	#	A_4
1	8	8	$A_4\#A_0C_0A_1C_1A_2A_3G_0C_2$	A_4	C_2
2	4	3	$A_2A_3G_0C_2A_4\#A_0C_0A_1C_1$	A_2	C_1
3	2	7	$A_1C_1A_2A_3G_0C_2A_4\#A_0C_0$	A_1	C_0
4	0	2	$A_0C_0A_1C_1A_2A_3G_0C_2A_4\#$	A_0	#
5	5	5	$A_3G_0C_2A_4\#A_0C_0A_1C_1A_2$	A_3	A_2
6	7	9	$C_2A_4\#A_0C_0A_1C_1A_2A_3G_0$	C_2	G_0
7	3	6	$C_1A_2A_3G_0C_2A_4\#A_0C_0A_1$	C_1	A_1
8	1	1	$C_0A_1C_1A_2A_3G_0C_2A_4\#A_0$	C_0	A_0
9	6	0	$G_0C_2A_4\#A_0C_0A_1C_1A_2A_3$	G_0	A_3

$S[\mathsf{SA}[6]] = S[7] = C_2$, it is the 1^{st} suffix starting with an 'C', $\mathsf{iSA}[\mathsf{SA}[6]+1] = 1$ and we have $\mathsf{BWT}[1] = C_2$, it is also the 1^{st} 'C' in the BWT and it corresponds to the same position, 7, in the sequence.

According to the property 7.5.1, we construct an array LF, such that for each position i in BWT, LF$[i]$ gives the position of the corresponding letter in F. The formula of the LF array is given in the next section (Definition 7.5.1), after having introduced the bit vectors. We note that the F column is the sorted BWT, thus we can obtain it directly from the BWT.

Then to recover the input sequence, we start from the right with the terminator symbol. We know that this symbol is at the end of the sequence, thus $S[n-2] = \mathsf{BWT}[0]$. Then we search for $S[n-2]$ in the F column, it occurs at position LF$[0]$. Then we know that the letter that precedes it is $S[n-3] = \mathsf{BWT}[\mathsf{LF}[0]]$. Now we search for $S[n-3]$ in the BWT, it occurs at position LF[LF$[0]$], and we obtain the preceding letter $S[n-4] = \mathsf{BWT}[\mathsf{LF}[\mathsf{LF}[0]]]$. We go on until we reach again the terminator symbol, meaning that the input sequence has been successfully retrieved.

Example 7.5.3 The arrays F, BWT and LF for the sequence
$S = \overset{0}{A_0}\overset{1}{C_0}\overset{2}{A_1}\overset{3}{C_1}\overset{4}{A_2}\overset{5}{A_3}\overset{6}{G_0}\overset{7}{C_2}\overset{8}{A_4}\overset{9}{\#}$.

i	F$[i]$	BWT$[i]$	LF$[i]$
0	#	A_4	1
1	A_4	C_2	6
2	A_2	C_1	7
3	A_1	C_0	8
4	A_0	#	0
5	A_3	A_2	2
6	C_2	G_0	9
7	C_1	A_1	3
8	C_0	A_0	4
9	G_0	A_3	5

7.5.2 Bit vectors

A bit vector (also known as *bitmap*, *bitset*, *bit string* or *bit array*), is a data structure storing only two possibles values, '1' or '0', so they can be stored in one bit. They can be compressed efficiently, although there is always a trade-off between the space compression and the time to answer queries. Once constructed, we can add to them some additional data structures to answer *rank* and *select* queries in constant time [123–125]. For a bit vector B of length n, $rank_1(i)$ (resp. $rank_0(i)$) is defined as the number of '1's (resp. '0's) in range $[0, i)$. The operation $select_1(i)$ (resp. $select_0(i)$) is the reverse, it answers at which position the i^{th} '1' (resp. '0') is in the bit vector.

By using bit vectors and their additional data structures for rank and select queries, we can search for a letter in the BWT in constant time. To this aim, we use a bit vector for each letter in the alphabet, $B_\alpha[i] = 1$ if $\mathsf{BWT}[i] = \alpha$ for every $i \in [0\,..\,n-1]$ and $\alpha \in \Sigma$.

Definition 7.5.1 According to property 7.5.1, we define the LF array as follow:
For all i, $0 \leq i < n$,
$\mathsf{LF}[i] = 1 + B_{\mathsf{BWT}[i]}.rank_1(1) + \sum_{\beta \in \Sigma, \beta < \mathsf{BWT}[i]} B_\beta.rank_1(n)$, if $\mathsf{BWT}[i] \neq \#$,
$\mathsf{LF}[i] = 0$ otherwise.

LF[i] gives the position in the F column of the letter $S[\mathsf{SA}[i] - 1]$ if $\mathsf{SA}[i] > 0$ or 0 otherwise. Please note that the F column is sorted thus the k^{th} letter c is at position $1 + k + \sum_{\beta \in \Sigma, \beta < c} B_\beta.rank_1(n)$.

Example of exact pattern matching with the BWT

Thus using the same approach as the reversing of the BWT we can search for a pattern P efficiently. As an example we search for the pattern P=GCA in $S = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{\mathrm{ACACAAGCA\#}}$ (see an illustration in Example 7.5.4). We search for the pattern from right to left:

- First we are looking at the whole table and we consider the last letter of the motif, 'A'. $B_A.rank_1(n) = 5$, we know that there are 5 occurrences of 'A', the red '1's in the illustration. With *select* we obtain their positions in the BWT, and with the LF array we obtain the positions in the BWT of the suffixes starting with 'A', the green numbers in the illustration. They are in the interval $[1, 6)$, these indices are in blue in the next step of the example.

- Now amongst the suffixes starting with 'A', we are looking for those preceded by 'C'. To do so we use $B_C.rank_1()$, at the two extremities of our interval of interest $[1, 6)$. $B_C.rank_1[1] = 0$ and $B_C.rank_1[6] = 3$, thus there are three occurrences of the pattern 'CA'. The LF array gives the interval of suffixes starting with 'CA', it is $[6, 9)$.

- Now amongst them we search for those preceded by 'G'. $B_G.rank_1[6] = 0$ and $B_G.rank_1[9] = 1$, thus there is only one occurrence of 'GCA'.

This way we can count the occurrences of a pattern P of size m in time $\mathcal{O}(m)$.

However outputting the positions can take much more time as we will need to count the steps until we reach the terminator symbol. Indeed, once we have found that there is only one occurrence of the pattern GCA, we only know that the suffix starting with GCA is at position 9 in the BWT. To know its position we have to continue backward until we reach the position 0 in the BWT that corresponds to the starting position of the input, and thus, by counting the number of steps to reach the beginning, we know the position of the pattern in the sequence.

The BWT is particularly space efficient for small alphabets; indeed for an alphabet of 4 letters (A, C, G, T) we need 4 bit vectors of size n. Thus without even considering any compression it will take $\frac{n}{2}$ bytes to store the BWT. Moreover these bit vectors are sparse, as in overall they contains exactly $n - 1$ '1' and $3n + 1$ '0', thus they can be compressed efficiently.

Example 7.5.4 Searching for the pattern GCA in the sequence $S = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{\mathrm{ACACAAGCA\#}}$, $B_A, B_C,$ and B_G are the bit vectors of the BWT.

Searching for 'A' in S

i	B_A	B_C	B_G	LF[i]
0	1	0	0	1
1	0	1	0	6
2	0	1	0	7
3	0	1	0	8
4	0	0	0	0
5	1	0	0	2
6	0	0	1	9
7	1	0	0	3
8	1	0	0	4
9	1	0	0	5

S=ACACAAGCA#

Searching for 'CA' in S

i	B_A	B_C	B_G	LF[i]
0	1	0	0	1
1	0	1	0	6
2	0	1	0	7
3	0	1	0	8
4	0	0	0	0
5	1	0	0	2
6	0	0	1	9
7	1	0	0	3
8	1	0	0	4
9	1	0	0	5

S=ACACAAGCA#

Searching for 'GCA' in S

i	B_A	B_C	B_G	LF[i]
0	1	0	0	1
1	0	1	0	6
2	0	1	0	7
3	0	1	0	8
4	0	0	0	0
5	1	0	0	2
6	0	0	1	9
7	1	0	0	3
8	1	0	0	4
9	1	0	0	5

S=ACACAAGCA#

The numbers in blue indicate the rows we are interested in . The red '1's indicate the rows of interest that are preceded by the letter we are looking for. The green numbers indicate their corresponding row in the F column.

7.5.3 FM-index

We have seen that the SA and the BWT are highly related, they are both obtained by sorting the suffixes of S. An idea introduced by Ferragina and Manzini in 2000 [107], was to use both of them at the same time. The structure, called the FM-index, can be used for pattern matching in the same way as the BWT, and it uses the SA to output the position of an occurrence in constant time. This structure is much more space efficient than the SA plus the LCP and it achieves the pattern matching in the same asymptotic time $\mathcal{O}(m + \#occ)$ for a pattern of size m. To achieve an interesting trade-off between space and time. Ferragina and Manzini proposed to sample the SA, by keeping only the value for the indices of the form $j\lfloor \log^{1+\epsilon} n \rfloor$, with $0 \leq j \leq \frac{n}{\lfloor \log^{1+\epsilon} n \rfloor}$ and ϵ a strictly positive constant. Then the LF-mapping property is used to obtain the value of a position that have been deleted. The idea is to traverse the text from right to left by using the LF array, until reaching a sampled position. The position of interest can be deduced from the retrieved position, by adding to it the number of steps that have been necessary to reach the sampled position. This solution is space efficient as the space complexity of the sampled suffix array is $o(\frac{n}{\log^{1+\epsilon} n})$, but it requires $\mathcal{O}(\log^{1+\epsilon} n)$ time to locate an occurrence of a pattern in the input sequence.

7.5.4 Applications to pattern matching in Bioinformatics

NGS also known as high-throughput sequencing technologies give a large number of reads, paired or not, and of different size, from a few dozen to several thousands of base pairs, depending on the technology. The reads obtained from a sequencing run are stored in FastQ [126] or BAM [37] files. There are two main ways to analyse these files: we can either assembly the genome *de novo*, or align (we also say map) the reads to a reference genome of the same specie. The read mapper algorithms are very important in NGS data analysis. Most of them are based on indexing data structures we have presented, to cite only a few, there are Bowtie [32, 33], BWA [31, 127] and SOAP [29, 36].

7.6 Summary of the different data structures

The different indexing data structures presented in this section are summarised in Table 7.1. The model of computation is the RAM with a word size of $\Theta(\log n)$ bits, with n the size of S, the input sequence. This way the space taken by an integer representing a position from 0 to n is constant. The column 'Counting occurrences' contains the complexity in time to output the number of occurrences of a pattern P of size m in S. The column 'Localisation' contains the complexity in time to locate the starting position in S of on occurrence of P after having counted its occurrences.

Data-structure	Space	Counting occurrences	Localisation
Suffix trie	$\mathcal{O}(n^2)$	$\mathcal{O}(m)$	$\mathcal{O}(1)$
Compact suffix tree	$\mathcal{O}(n)$	$\mathcal{O}(m)$	$\mathcal{O}(1)$
Suffix array	$\mathcal{O}(n)$	$\mathcal{O}(m \log n)$	$\mathcal{O}(1)$
Enhanced suffix array	$\mathcal{O}(n)$	$\mathcal{O}(m)$	$\mathcal{O}(1)$
FM-index	$nH_k(S) + o(\frac{n}{\log^{1+\epsilon} n})$	$\mathcal{O}(m)$	$\mathcal{O}(\log^{1+\epsilon} n)$

Table 7.1: Complexities in space and query time for indexing data structure when considering a fixed size alphabet. The model of computation is the RAM with a word size of $\Theta(\log n)$ bits.

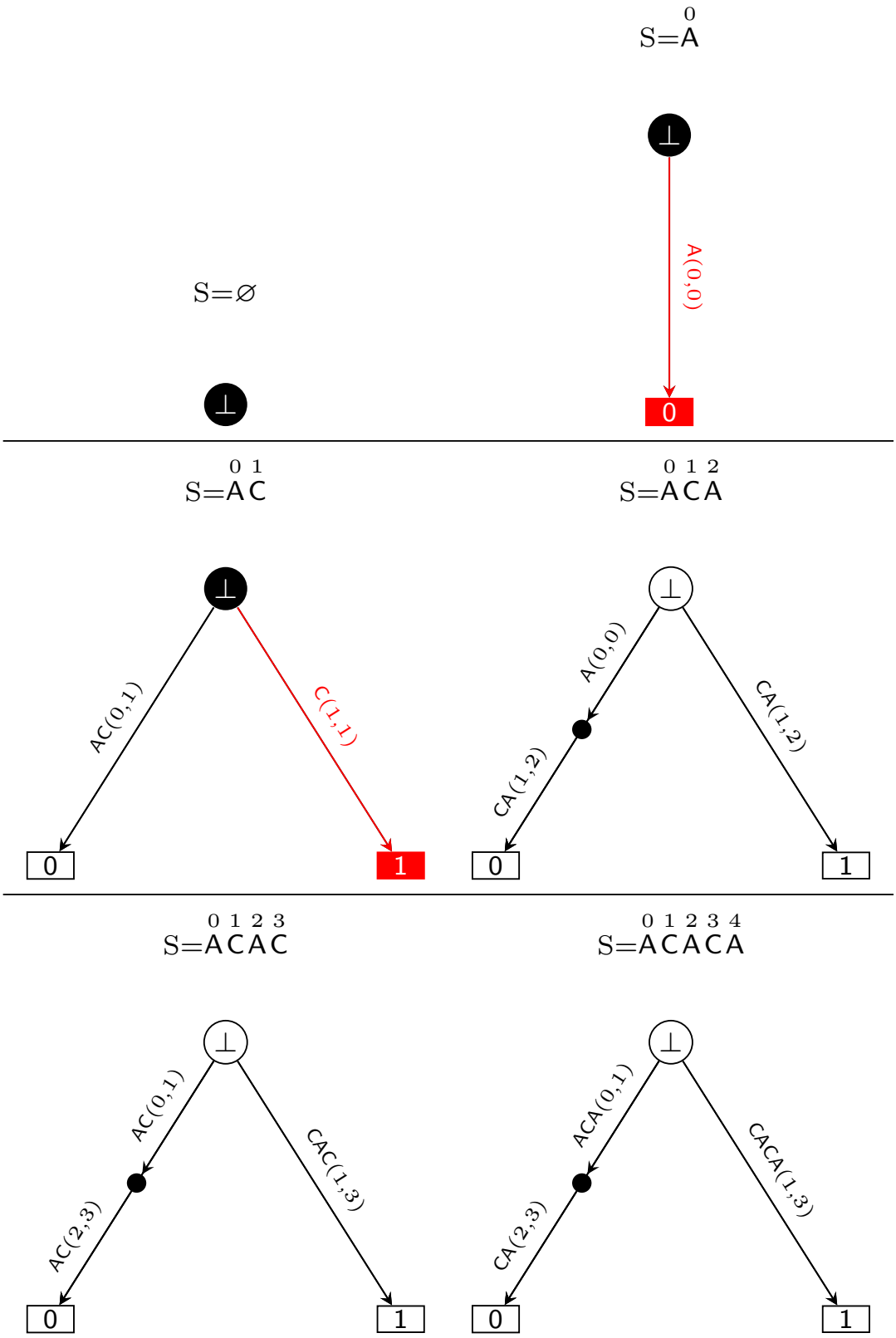


Figure 7.4: Illustration of the Ukkonen construction algorithm step by step for the sequence S=ACACAAGCA#. The active point is represented in black. New nodes, leaves and edges are in red.

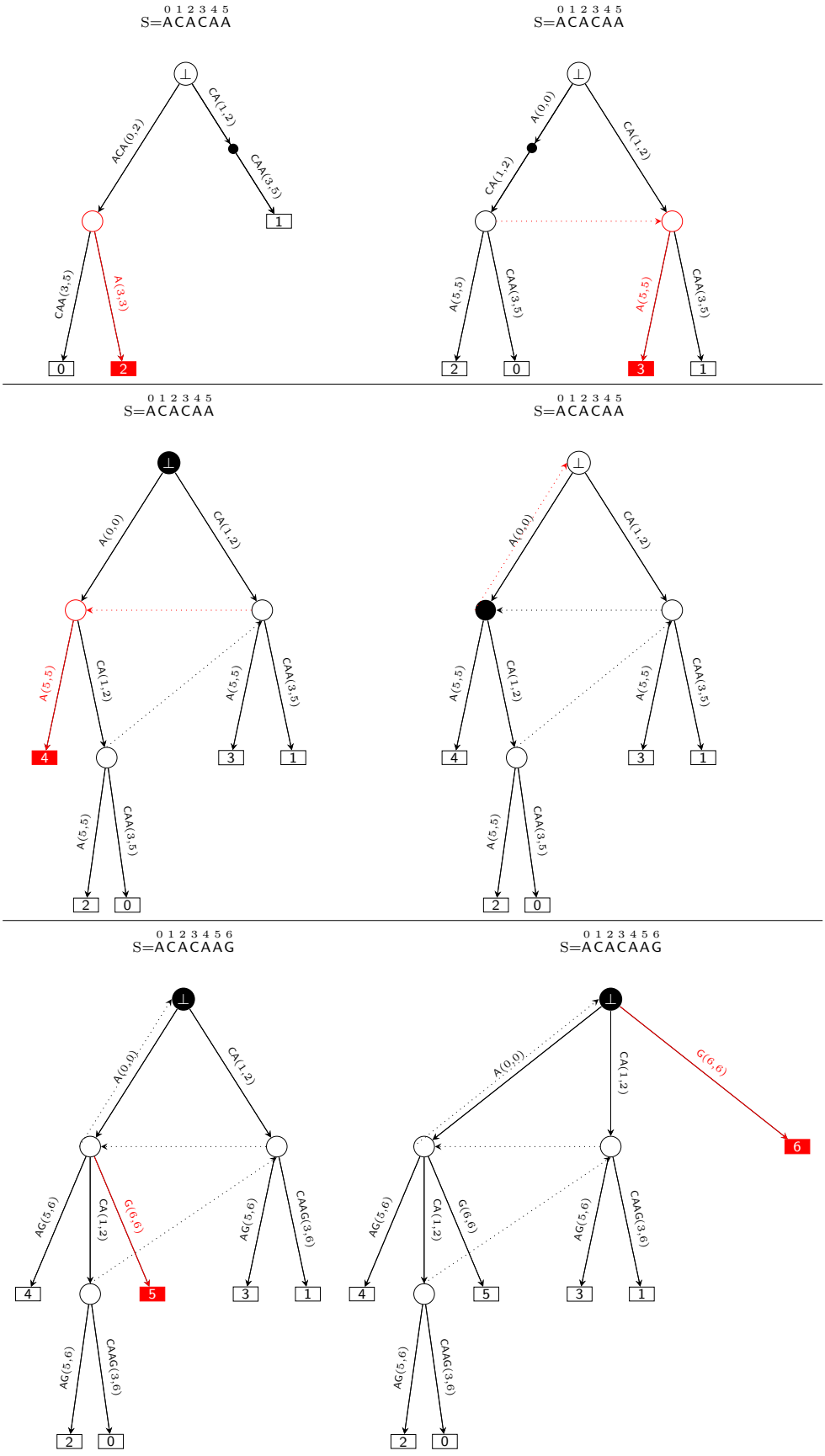


Figure 7.5: The continuation of the illustration of the Ukkonen construction algorithm step by step for the sequence S=ACACAAGCA#.

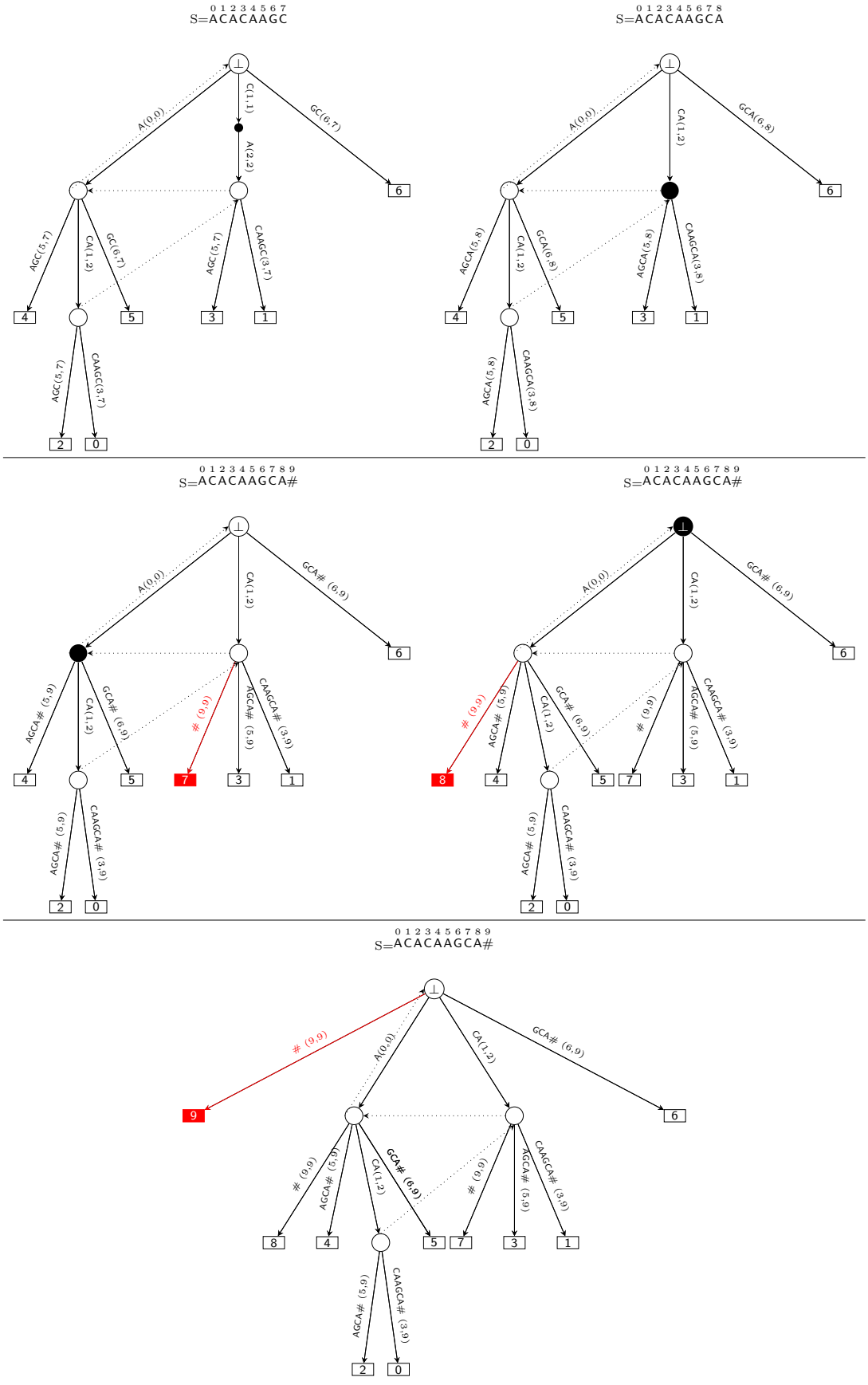


Figure 7.6: The end of the illustration of the Ukkonen construction algorithm step by step for the sequence S=ACACAAGCA#.

8

Linear time and space computations of minimal absent words

Contents

8.1 Problem and previous approaches

Now that we have presented all the necessary definitions and data structures, we present different algorithms to compute minimal absent words (see Definition 6.2.1) that I have devised and implemented during my PhD. The problem we want

to solve is the following.

Problem 1: Computation of Minimal Absent Words
Input: A sequence S of length n on a fixed-size alphabet Σ.
Output: For every minimal absent words w of S, one tuple $< a, (i, j) >$, such that $w = a.S[i\,..\,j]$.

There already exists $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithms for computing all minimal absent words based on the construction of suffix automata [73]. An alternative $\mathcal{O}(n)$-time solution for finding minimal absent words of length at most ℓ, with $\ell = \mathcal{O}(1)$, based on the construction of tries of bounded-length factors was presented in [84].

A drawback of these approaches, in practical terms, is that the construction of suffix automata (or of tries) may have a large memory footprint. Due to this, an important problem is to be able to compute the minimal absent words of a sequence without the use of data structures such as the suffix automaton or the suffix tree.

To this end, the computation of minimal absent words based on the construction of suffix arrays was considered in [76]; although fast in practice, the worst-case runtime of this algorithm is $\mathcal{O}(n^2)$. Then Fukae et al. [128] proposed a linear time and space algorithm to compute the set of minimal absent words. They claimed that their algorithm is fast and memory-efficient, but they did not provide an implementation. Alternatively, one could make use of the succinct representations of the bidirectional BWT, recently presented in [129], to compute all minimal absent words in time $\mathcal{O}(n)$ with a small memory footprint. However, an implementation of these representations was not made available, and it is also unlikely that such an implementation will outperform an $\mathcal{O}(n)$-time algorithm based on the construction of suffix arrays.

We first present the algorithm MAW [130] that computes all minimal absent words of a sequence of length n based on the suffix arrays and whose implementation is available. It is shown to be more efficient than the existing tools at the time, both in terms of speed and memory. Then we introduce the algorithm pMAW [99], a new algorithm solving the same problem in linear time and space but with the additional property that it can be executed in parallel. By excluding the indexing data-structure construction time, the implementation achieves near-optimal speed-ups. Finally, we explain how the first algorithm can be slightly modify into em-MAW [100] an algorithm compatible with external memory computation. We have provided an implementation that is slightly slower but that consumes much less internal memory.

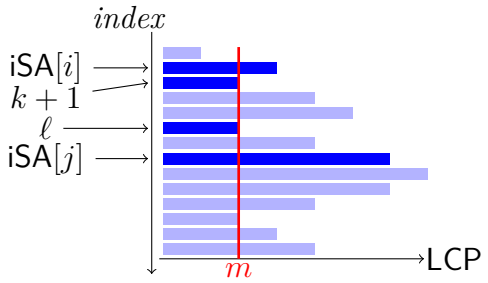


Figure 8.1: Illustration of Lemma 8.2.1. (i, j, w) is a maximal repeated pair and $w = S[\mathsf{SA}[k]\,..\,\mathsf{SA}[k] + \mathsf{LCP}[k+1] - 1] = S[\mathsf{SA}[\ell]\,..\,\mathsf{SA}[\ell] + \mathsf{LCP}[\ell] - 1]$

8.2 MAW

In this section, we present algorithm MAW [130], an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm for finding all minimal absent words in a sequence of length n using arrays SA and LCP.

As we have seen in Lemma 6.2.1 minimal absent words are closely related to maximal repeated pairs (see Definition 6.2.2). Thus the main idea of minimal absent words computation is to compute the set of letters that occur right before and after each maximal repeated pair and see if they can form a minimal absent word.

Lemma 8.2.1. *Let (i, j, w) be a right maximal repeated pair then there exists $0 \leq k < \ell < n$, such that: $S[i\,..\,i + |w|] = S[SA[k]\,..\,SA[k] + LCP[k+1]]$ and $S[j\,..\,j + |w|] = S[SA[\ell]\,..\,SA[\ell] + LCP[\ell]]$.*

Proof. Let (i, j, w) be a right maximal repeated pair with $\mathsf{iSA}[i] < \mathsf{iSA}[j]$; if this is not the case we consider the triple (j, i, w). The two suffixes starting at positions iSA[i] and iSA[j] share the same prefix of length $|w|$ and then they differ. Thus, we have $\mathsf{lcp}(\mathsf{iSA}[i], \mathsf{iSA}[j]) = |w|$, according to remark 7.4.1, there is at least one $m \in (\mathsf{iSA}[i], \mathsf{iSA}[j]]$ such that $\mathsf{LCP}[m] = |w|$. We choose ℓ to be the largest index in $(\mathsf{iSA}[i], \mathsf{iSA}[j]]$, such that $\mathsf{LCP}[\ell] = |w|$, and $k+1$ to be the smallest; they can be equal. Thus $\mathsf{lcp}(k, \mathsf{iSA}[i]) > |w|$, so $S[\mathsf{SA}[k]\,..\,\mathsf{SA}[k] + |w|] = S[i\,..\,i + |w|]$. Similarly, $\mathsf{lcp}(\ell, \mathsf{iSA}[j]) > |w|$, so $S[\mathsf{SA}[\ell]\,..\,\mathsf{SA}[\ell] + |w|] = S[j\,..\,j + |w|]$. For an illustration inspect Fig. 8.1. □

By Lemma 8.2.1, we can focus onto the following $2n$ factors to consider all right maximal repeated pairs:

- $F_{2i} = S[\mathsf{SA}[i] \ldots \mathsf{SA}[i] + \mathsf{LCP}[i]]$, with $i \in [0 : n-1]$.
- $F_{2i+1} = S[\mathsf{SA}[i] \ldots \mathsf{SA}[i] + \mathsf{LCP}[i+1]]$, with $i \in [0 : n-1]$.

Lemma 8.2.2. *For all* $0 \leq i \neq j \leq n-1$, $F_{2i} \neq F_{2j}$ *and* $F_{2i+1} \neq F_{2j+1}$

Proof. Without loss of generality we can suppose that $i < j$. By definition of LCP, $S[\mathsf{SA}[j-1] \dots \mathsf{SA}[j-1] + \mathsf{LCP}[j]] < S[\mathsf{SA}[j] \dots \mathsf{SA}[j] + \mathsf{LCP}[j]]$. The suffixes are ordered in the SA thus $i < j$ implies $S[\mathsf{SA}[i] \dots \mathsf{SA}[i] + \mathsf{LCP}[j]] \leq S[\mathsf{SA}[j-1] \dots \mathsf{SA}[j-1] + \mathsf{LCP}[j]]$. Consequently $F_{2i} < F_{2j}$.

Similarly, we have $S[\mathsf{SA}[i] \dots \mathsf{SA}[i]+\mathsf{LCP}[i+1]] < S[\mathsf{SA}[i+1] \dots \mathsf{SA}[i+1]+\mathsf{LCP}[i+1]]$. And $i < j$ implies $S[\mathsf{SA}[i+1] \dots \mathsf{SA}[i+1] + \mathsf{LCP}[i+1]] \leq S[\mathsf{SA}[j] \dots \mathsf{SA}[j] + \mathsf{LCP}[j+1]]$. Consequently $F_{2i+1} < F_{2j+1}$. □

These $2n$ factors are sufficient to compute the set of minimal absent words, but there is redundancy because some of them are equals. This is proved in the following Lemma.

Lemma 8.2.3. *For* $0 \leq i \neq j \leq 2n-1$, $F_i = F_j$ *if and only if there exist* $0 \leq a < b \leq n-1$ *such that* $i = 2a$, $j = 2b+1$, $LCP[a] = LCP[b+1]$, *and, for each* ℓ *in* $[a+1 : b]$, $LCP[\ell] > LCP[a]$.

Proof. According to lemma 8.2.2, $F_i = F_j$ is possible only if i and j do not have the same parity. Thus, without loss of generality we take i even, $i = 2a$, and j odd, $j = 2b+1$; then, $y[\mathsf{SA}[a] \,..\, \mathsf{SA}[a] + \mathsf{LCP}[a]] = y[\mathsf{SA}[b] \,..\, \mathsf{SA}[b] + \mathsf{LCP}[b+1]]$. By definition, $y[\mathsf{SA}[a] \,..\, \mathsf{SA}[a] + \mathsf{LCP}[a]] > y[\mathsf{SA}[a-1] \,..\, \mathsf{SA}[a-1] + \mathsf{LCP}[a]]$, thus $b > a$. Moreover we have $\mathsf{LCP}[a] = \mathsf{LCP}[b+1]$ so, for each ℓ in $[a+1 : b]$, $\mathsf{LCP}[\ell] > \mathsf{LCP}[a]$.

Reciprocally, if there are $a < b$ such that $\mathsf{LCP}[a] = \mathsf{LCP}[b+1]$, and, for each ℓ in $[a+1 : b]$, $\mathsf{LCP}[\ell] > \mathsf{LCP}[a]$, then, $y[\mathsf{SA}[a] \,..\, \mathsf{SA}[a] + \mathsf{LCP}[a]] = y[\mathsf{SA}[b] \,..\, \mathsf{SA}[b] + \mathsf{LCP}[b+1]]$ so $F_{2a} = F_{2b+1}$. □

For each F_j, with $j \in [0 : 2n-1]$, we denote by:

- $B_1[j]$ the set of letters that occur right before the occurrences of F_j.
- $B_2[j]$ the set of letters that occur right before the occurrences of the longest proper prefix of F_j.

Remark 8.2.1 Note that for every $j \in [0 : 2n-1]$, $B_2[j] \subset B_1[j]$.

Lemma 8.2.4. *awb is a minimal absent word of* S, *with* a, b *letters and* w *a word, if and only if there exists* j *such that* $a \in B_2[j] \setminus B_1[j]$ *and* $wb = F_j$.

Proof. If awb is a minimal absent word then by Lemma 6.2.1, there are i and j with $\mathsf{iSA}[i] < \mathsf{iSA}[j]$ such that (i, j, w) is a maximal repeated pair. One position, i or j, is a starting position of wb.

Case 1: If j is a starting position of wb then by Lemma 8.2.1, there exists ℓ such that $\mathsf{SA}[\ell]$ is a starting position of wb and $\mathsf{LCP}[\ell] = |w|$. Thus $wb = F_{2\ell}$, aw occurs in S but awb does not, and a is in $B_2(2\ell) \setminus B_1(2\ell)$.

Case 2: If i is a starting position of wb then by Lemma 8.2.1, there exists k such that $\mathsf{SA}[k]$ is a starting position of wb and $\mathsf{LCP}[k+1] = |w|$. Thus $wb = F_{2k+1}$, aw occurs in S but awb does not, and a is in $B_2(2k+1) \setminus B_1(2k+1)$.

Reciprocally, if there is an index j and a letter a such that a is in $B_2(j) \setminus B_1(j)$, we denote by w the longest proper prefix of F_j. Then aw and F_j occur in S but aF_j does not. Consequently aF_j is a minimal absent word of S. □

By Lemma 8.2.4, the difference between $B_1[j]$ and $B_2[j]$, for all j in $[0 : 2n-1]$, gives us all the minimal absent words of S.

Thus the important point is to compute these sets of letters efficiently. To do so, we visit twice arrays SA and LCP. While iterating over these arrays, we maintain another array denoted by Pref, such that, at the end of each iteration i, the ℓ^{th} element of Pref stores the set of letters we have encountered before the prefix of length ℓ of $S[\mathsf{SA}[i] \,..\, n-1]$. Array Pref consists of $\max_{k\in[0:n-1]} \mathsf{LCP}[k] + 1$ elements, where each element is a bit vector of length σ, the size of the alphabet. Thus it is of size $\mathcal{O}(\sigma n)$.

During the first pass, we visit arrays SA and LCP from top to bottom. For each $i \in [0 : n-1]$, we store in positions $2i$ and $2i+1$ of B_1 (resp. B_2) the set of letters that immediately precedes the occurrences of F_{2i} and F_{2i+1} (resp. their longest proper prefixes) whose starting positions appear before position i in SA. During the second pass, we go bottom up to complete the sets, which are already stored, with the letters preceding the occurrences whose starting positions appear after position i in SA. In order to be efficient, we maintain a stack structure, denoted by $\mathsf{LifoLCP}$, to store the LCP values of the factors that are prefixes of the factor we are currently visiting.

8.2.1 Top-down Pass

Each iteration of the top-down pass consists of two steps. In the first step, we visit $\mathsf{LifoLCP}$ from the top and for each LCP value read we set to zero the corresponding element of Pref; then we remove this value from the stack. We stop when we reach a value smaller or equal to $\mathsf{LCP}[i]$. We do this as the corresponding factors are not prefixes of $S[\mathsf{SA}[i] \,..\, n-1]$, nor will they be prefixes in the remaining

suffixes. We push at most one value onto the stack LifoLCP per iteration, so, in total, there are n times we will set an element of Pref to zero. Thus in overall this step requires time and space $\mathcal{O}(n\sigma)$.

For the second step, we update the elements corresponding to factors in the suffix array with a LCP value smaller than LCP[i]. To do so, we visit the stack LifoLCP top-down and, for each LCP value ℓ read, we add the letter $S[\mathsf{SA}[i]-1]$ to Pref[ℓ] until we reach a value whose element already contains it. This ensures that, for each value read, the corresponding element of Pref has no more than σ letters added. We have shown above that in overall we set an element of Pref to zero at most n times. Thus filling all the elements of Pref during the Top-down pass requires $\mathcal{O}(n\sigma)$ time and space. For an example, see Table 8.1.

```
Function Top-Down-Pass (S, n, SA, LCP, B1, B2, σ)
    Pref[0..max_{i∈[0:n−1]} LCP[i]][0..σ−1] ← 0;
    LifoLCP.push(0);
    foreach i ∈ [0 : n−1] do
        if i>0 and LCP[i] < LCP[i−1] then
            while LifoLCP.top() > LCP[i] do
                proxa ← LifoLCP.pop();
                Pref[proxa][0..σ−1] ← 0;
            if LifoLCP.top() < LCP[i] then
                Pref[LCP[i]] ← Pref[proxa];
            B1[2i−1] ← Pref[proxa]; B2[2i−1] ← Pref[LCP[i]];
        if SA[i] > 0 then
            u ← S[SA[i]−1]; value ← LifoLCP.top();
            while Pref[value][u] = 0 do
                Pref[value][u] ← 1; value ← LifoLCP.next();
            Pref[LCP[i]][u] ← 1;
            B1[2i][u] ← 1; B1[2i+1][u] ← 1;
            B2[2i][u] ← 1; B2[2i+1][u] ← 1;
        if i>0 and LCP[i] > 0 and SA[i−1] > 0 then
            v ← S[SA[i−1]−1];
            Pref[LCP[i]][v] ← 1 ;
        B2[2i] ← Pref[LCP[i]];
        if LifoLCP.top() ≠ LCP[i] then LifoLCP.push(LCP[i]);
```

j	B_1	B_2
0	∅	∅
1	∅	∅
2	A	A
3	A	A
4	C	A,C
5	A,C	A,C
6	C	A,C
7	A,C	A,C
8	C	A,C
9	C	C
10	A	A,C
11	A	A
12	A	A
13	A	A,C
14	A	A,C

(a)

i	LCP	SA	Factor	Pref[0]	Pref[1]	Pref[2]	Pref[3]	Pref[4]
0	0	0	A	∅	∅	∅	∅	∅
			A A					
1	1	1	A C	A	A	∅	∅	∅
			A C A C A					
2	4	3	A C A C C	A,C	A,C	∅	∅	A,C
			A C A					
3	2	5	A C C	A,C	A,C	A,C	∅	∅
			A					
4	0	7	C	A,C	∅	∅	∅	∅
			C #					
5	1	2	C A	A,C	A,C	∅	∅	∅
			C A C A					
6	3	4	C A C C	A,C	A,C	∅	A	∅
			C A					
7	1	6	C C	A,C	A,C	∅	∅	∅

(b)

i	LifoLCP
0	0
1	$0 \rightarrow \mathsf{LCP}[1] = 1$
2	$0 \rightarrow \mathsf{LCP}[1] = 1 \rightarrow \mathsf{LCP}[2] = 4$
3	$0 \rightarrow \mathsf{LCP}[1] = 1 \rightarrow \mathsf{LCP}[3] = 2$
4	0
5	$0 \rightarrow \mathsf{LCP}[5] = 1$
6	$0 \rightarrow \mathsf{LCP}[5] = 1 \rightarrow \mathsf{LCP}[6] = 3$
7	$0 \rightarrow \mathsf{LCP}[5] = 1$

(c)

Table 8.1: Illustration of the top-down pass of algorithm MAW for the word $S =$ `AACACACC` over an alphabet of size 2. (a) Arrays B_1 and B_2 obtained after the top-down pass; (b) Elements of array Pref at the end of each iteration of the top-down pass. Factors F_j are in orange and red; their longest proper prefixes are in orange only; (c) The stack LifoLCP at the end of each iteration.

8.2.2 Bottom-up Pass

Intuitively, the idea behind the bottom-up pass is the same as in the top-down pass except that in this instance, as we start from the bottom, the suffix $S[\mathsf{SA}[i]\,..\,n-1]$ can share more than its prefix of length $\mathsf{LCP}[i]$ with the previous suffixes in SA. Therefore we may need the elements of Pref that correspond to factors with a LCP value greater than $\mathsf{LCP}[i]$ to correctly compute the arrays B_1 and B_2. To achieve this, we maintain another stack LifoRem to copy the values from LifoLCP that are greater than $\mathsf{LCP}[i]$. This extra stack allows us to keep in LifoLCP only values that are smaller or equal to $\mathsf{LCP}[i]$ without losing the additional information we need to correctly compute B_1 and B_2. At the end of the iteration, we will set to zero each element corresponding to a value in LifoRem and empty the stack. Thus to set an

element of Pref to zero requires two operations more than in the first pass. As we consider at most n values, this step requires in overall time and space $\mathcal{O}(n\sigma)$.

Another difference between the top-down and bottom-up passes is that in order to retain the information computed in the first pass, the second step is performed for each letter in $B_1[2i]$. As for each LCP value read we still add a letter only if it is not already contained in the corresponding element of Pref. No more than σ letters are added. Thus this step requires in overall time and space $\mathcal{O}(n\sigma)$. For an example, see Table 8.2.

```
Function Bottom-Up-Pass(n, SA, LCP, B1, B2, Σ, σ)
    Pref[0 .. max_{i∈[0:n−1]} LCP[i]][0 .. σ − 1] ← 0;
    LifoLCP.push(0);
    foreach i ∈ [n − 1 : 0] do
        proxa ← LCP[i] + 1; proxb ← 1;
        if i<n-1 and LCP[i] < LCP[i + 1] then
            while LifoLCP.top() > LCP[i] do
                proxa ← LifoLCP.pop();
                LifoRem.push(proxa);
            if LifoLCP.top() < LCP[i] then
                Pref[LCP[i]] ← Pref[proxa]
        foreach k ∈ Σ : B1[2i][k] = 1 do
            value ← LifoLCP.top();
            while Pref[value][k] = 0 do
                Pref[value][k] ← 1; value ← LifoLCP.next();
            Pref[LCP[i]][k] ← 1;
        B2[2i]     ← B2[2i]     bit-or Pref[LCP[i]];
        B2[2i + 1]← B2[2i + 1] bit-or Pref[LCP[i + 1]];
        B1[2i + 1]← B1[2i + 1] bit-or Pref[proxb];

        proxb ← proxa;
        B1[2i] ← B1[2i] bit-or Pref[proxa];
        while LifoRem not empty do
            value ← LifoRem.pop(); Pref[value][0 .. σ − 1] ← 0;
        if LifoLCP.top() ≠ LCP[i] then LifoLCP.push(LCP[i]);
```

j	B_1	B_2
0	A,C	A,C
1	∅	A,C
2	A,C	A,C
3	A	A,C
4	C	A,C
5	A,C	A,C
6	C	A,C
7	A,C	A,C
8	A,C	A,C
9	C	A,C
10	A	A,C
11	A	A
12	A	A
13	A	A,C
14	A	A,C

(a)

i	LCP	SA	Factor	Pref[0]	Pref[1]	Pref[2]	Pref[3]	Pref[4]
7	1	6	C C	A	A	∅	∅	∅
			C A					
6	3	4	C A C C	A	A	∅	A	∅
			C A C A					
5	1	2	C A	A	A	∅	∅	∅
			C #					
4	0	7	C	A,C	∅	∅	∅	∅
			A					
3	2	5	A C C	A,C	∅	C	∅	∅
			A C A					
2	4	3	A C A C C	A,C	∅	C	∅	C
			A C A C A					
1	1	1	A C	A,C	A,C	∅	∅	∅
			A A					
0	0	0	A	A,C	∅	∅	∅	∅

(b)

i	LifoLCP
7	$0 \rightarrow \mathsf{LCP}[7] = 1$
6	$0 \rightarrow \mathsf{LCP}[7] = 1 \rightarrow \mathsf{LCP}[6] = 3$
5	$0 \rightarrow \mathsf{LCP}[7] = 1$
4	0
3	$0 \rightarrow \mathsf{LCP}[3] = 2$
2	$0 \rightarrow \mathsf{LCP}[3] = 2 \rightarrow \mathsf{LCP}[2] = 4$
1	$0 \rightarrow \mathsf{LCP}[1] = 1$
0	0

(c)

Table 8.2: Illustration of the bottom-up pass of algorithm MAW for the word $S =$ AACACACC over an alphabet of size 2. (a) Arrays B_1 and B_2 obtained after the bottom-up pass; (b) Elements of array Pref at the end of each iteration of the bottom-up pass. Factors F_j are in orange and red; their longest proper prefixes are in orange only; (c) The stack LifoLCP at the end of each iteration.

8.2.3 Deducing the set of minimal absent words

Once we have computed arrays B_1 and B_2, we need to compare each element. If the difference is not empty, by Lemma 8.2.4, we can construct some minimal absent words. For an example, see Table 8.3. However we observe that this computation may result in reporting duplicates when there exist i and j such that $F_i = F_j$. To avoid duplicates, we first build a bit-vector Dup of size n, that contains '1' at position k if and only if there exists an $\ell < k$ such that $\mathsf{LCP}[\ell] = \mathsf{LCP}[k]$ and $F[2\ell] = F[2k+1]$. We formalise this procedure in Function Avoid duplicates, that runs in linear time.

j	B_1	B_2	Factor					Tuple representation of Minimal absent words
0	A,C	A,C	A					
1	∅	A,C	A	A				<A,(0,1)>, <C,(0,1)>
2	A,C	A,C	A	C				
3	A	A,C	A	C	A	C	A	<C,(1,5)>
4	C	A,C	A	C	A	C	C	<A,(3,7)>
5	A,C	A,C	A	C	A			
6	C	A,C	A	C	C			<A,(5,7)>
7	A,C	A,C	A					
8	A,C	A,C	C					
9	C	A,C	C	#				We do not consider this row as it corresponds to the end of the sequence S
10	A	A,C	C	A				<C,(2,3)>
11	A	A	C	A	C	A		
12	A	A	C	A	C	C		
13	A	A,C	C	A				This is a duplicate of row 10 so we ignore it
14	A	A,C	C	C				<C,(6,7)>

Table 8.3: Minimal absent words of word $S =$ AACACACC; we find seven minimal absent words {AAA, AACACC, AACC, CAA, CACACA, CCA, CCC}

Function Avoid duplicates *(n, LCP)*
Stack lifo_lcp; int lcp=0; int mem;
lifo_lcp.push(lcp); Dup[0 . . $n-1$]←0;
foreach $i \in [0 : n-1]$ **do**
lcp←lifo_lcp.pop();
while lifo_lcp is not empty and lcp>LCP[i] **do**
mem←lifo_lcp.pop;
if mem=LCP[i] **then** Dup[i]=1;
lcp←mem
lifo_lcp.push(lcp); lcp←LCP[i];
lifo_lcp.push(lcp);

Then according to Lemma 8.2.3, we can avoid the duplicates by not considering the indices j such that j is odd and $\mathsf{Dup}[(j+1)/2] = 1$. Finally, to compute and report all minimal absent words exactly once, we compute the difference $B_2(j) \setminus B_1(j)$, for all $j \in [0 : 2n-1]$, such that j is even or $\mathsf{Dup}[(j+1)/2] = 0$.

To report each minimal absent words in constant time, we represent them as a tuple $< a, (i, j) >$. Where for some word x of length $m \geq 2$, that is a minimal absent word of S, the following holds: $x[0] = a$ and $x[1 . . m-1] = S[i . . j]$. Lemma 8.2.1 ensures exhaustivity, therefore we obtain the following theorem.

Theorem 8.2.5. *Algorithm MAW solves problem Computation of minimal absent words in time and space* $\mathcal{O}(n)$.

8.2.4 Results

We implemented algorithm MAW as a program to compute all minimal absent words of a given sequence. The program was implemented in the C programming language and developed under GNU/Linux operating system. It takes as input arguments a file in (Multi)FASTA format and the minimal and maximal length of minimal absent words to be outputted; and then produces a file with all minimal absent words of length within this range. The implementation is distributed under the GNU General Public License (GPL), and it is available at `http://github.com/solonas13/maw`, which is set up for maintaining the source code and the man-page documentation. The experiments were conducted on a Desktop PC using one core of Intel Xeon E5540 CPU at 2.5 GHz and 32GB of main memory under 64-bit GNU/Linux. We considered the genomes of eleven bacteria and four case-study eukaryotes (Table 8.4), all obtained from the NCBI database (`ftp://ftp.ncbi.nih.gov/genomes/`).

Species	Abbreviation	Genome reference
Bacteria		
Bacillus anthracis strain Ames	Ba	NC003997
Bacillus subtilis strain 168	Bs	NC000964
Escherichia coli strain K-12 substrain MG1655	Ec	NC000913
Haemophilus influenzae strain Rd KW20	Hi	NC000907
Helicobacter pylori strain 26695	Hp	NC000915
Lactobacillus casei strain BL23	Lc	NC010999
Lactococcus lactis strain Il1403	Ll	NC002662
Mycoplasma genitalium strain G37	Mg	NC000908
Staphylococcus aureus strain N315	Sa	NC002745
Streptococcus pneumoniae strain CGSP14	Sp	NC010582
Xanthomonas campestris strain 8004	Xc	NC007086
Eukaryotes		
Arabidopsis thaliana (thale cress)	At	AGI release 7.2
Drosophila melanogaster (fruit fly)	Dm	FlyBase release 5
Homo sapiens (human)	Hs	build 38
Mus musculus (mouse)	Mm	build 38

Table 8.4: Species selected for this work with reference to the respective abbreviation and identification of genome sequence data by accession number for bacteria or genome assembly project for eukaryotes.

To test the correctness of our implementation, we compared it against the implementation of Pinho et al. [76], which we denote here by PFG. In particular, we counted the number of minimal absent words, for lengths 11, 14, 17, and 24, in the genomes of the eleven bacteria listed in Table 8.4. We considered only the $5' \rightarrow 3'$ DNA strand. Table 8.5 depicts the number of minimal absent words in these sequences. We denote by M_{11}, M_{14}, M_{17}, and M_{24} the size of the resulting

sets of minimal absent words for lengths 11, 14, 17, and 24 respectively. Identical number of minimal absent words for these lengths were also reported by PFG, suggesting that our implementation is correct.

Species	Genome size (bp)	M_{11}	M_{14}	M_{17}	M_{24}
Ba	5,227,293	1,113,398	1,001,357	32,432	46
Bs	4,214,630	951,273	1,703,309	86,372	226
Ec	4,639,675	1,072,074	1,125,653	36,395	247
Hi	1,830,023	722,860	294,353	12,158	91
Hp	1,667,825	564,308	336,122	19,276	75
Lc	3,079,196	1,126,363	502,861	13,083	246
Ll	2,365,589	764,006	507,490	25,667	183
Mg	1,664,957	246,342	66,324	2,737	28
Sa	2,814,816	755,483	704,147	32,054	138
Sp	2,209,198	904,815	327,713	10,390	234
Xc	5,148,708	804,034	1,746,214	179,346	633

Table 8.5: Number of minimal absent words of lengths 11, 14, 17, and 24 in the genomes of eleven bacteria.

To evaluate the efficiency of our implementation, we compared it against the corresponding performance of PFG, which is currently the fastest available implementation for computing minimal absent words. Notice that this evaluation depends heavily on the suffix array construction implementation used; and that PFG uses a less optimised implementation for this construction than the one used by MAW. We computed all minimal absent words for each chromosome sequence of the genomes of the four eukaryotes listed in Table 8.4. We considered both the $5' \rightarrow 3'$ and the $3' \rightarrow 5'$ DNA strands. Tables 8.6 and 8.7 depict elapsed-time comparisons of MAW and PFG. MAW scales linearly and is the fastest in all cases. It accelerates the computations by more than a factor of 2, when the length of the sequences grows, compared to PFG. MAW also reduces the memory requirements by a factor of 5 compared to PFG. The maximum allocated memory (per task) was 6GB for MAW and 30GB for PFG.

Chromosome	Size (bp)	MAW (s)	PFG (s)
1	30,427,671	40.20	51.90
2	19,698,289	25.86	32.94
3	23,459,830	30.84	42.30
4	18,585,056	24.65	31.42
5	26,975,502	35.38	48.91

(a) At

Chromosome	Size (bp)	MAW (s)	PFG (s)
2L	23,011,544	30.01	40.85
2R	21,146,708	27.52	38.38
3L	24,543,557	32.00	45.13
3R	27,905,053	36.44	48.36
X	22,422,827	29.38	40.09

(b) Dm

Table 8.6: Elapsed-time comparison of MAW and PFG for computing all minimal absent words in the genome of *Arabidopsis thaliana* and *Drosophila melanogaster*.

Chromosome	Size (bp)	MAW (s)	PFG (s)
1	248,956,422	426.39	972.52
2	242,193,529	423.19	772.89
3	198,295,559	353.60	645.45
4	190,214,555	339.02	616.26
5	181,538,259	342.53	577.05
6	170,805,979	299.72	538.34
7	159,345,973	305.26	491.32
8	145,138,636	254.17	437.18
9	138,394,717	235.14	356.08
10	133,797,422	235.38	392.45
11	135,086,622	236.80	379.15
12	133,275,309	235.14	390.46
13	114,364,328	191.64	269.52
14	107,043,718	178.00	240.93
15	101,991,189	167.89	222.98
16	90,338,345	153.07	198.49
17	83,257,441	144.32	207.02
18	80,373,285	137.68	199.44
19	58,617,616	100.95	126.82
20	64,444,167	109.80	144.83
21	46,709,983	74.65	74.65
22	50,818,468	80.49	73.34
X	156,040,895	275.14	457.2
Y	57,227,415	82.85	62.34

(a) Hs

Chromosome	Size (bp)	MAW (s)	PFG (s)
1	197,195,432	340.59	599.86
2	181,748,087	316.17	578.2
3	159,599,783	274.46	506.73
4	155,630,120	266.67	473.97
5	152,537,259	260.50	424.24
6	149,517,037	256.36	455.11
7	152,524,553	257.65	413.37
8	131,738,871	223.09	344.92
9	124,076,172	210.37	334.25
10	129,993,255	222.36	363.34
11	121,843,856	208.55	324.54
12	121,257,530	205.09	324.79
13	120,284,312	204.80	314.56
14	125,194,864	212.59	336.49
15	103,494,974	175.21	265.92
16	98,319,150	166.10	249.03
17	95,272,651	160.70	232.79
18	90,772,031	153.40	223.56
19	61,342,430	101.89	125.85
X	166,650,296	282.21	503.98
Y	91,744,698	141.79	251

(b) Mm

Table 8.7: Elapsed-time comparison of MAW and PFG for computing all minimal absent words in the genome of *Homo Sapiens* and *Mus musculus*.

To further evaluate the efficiency of our implementation, we compared it against the corresponding performance of PFG using synthetic data. As basic dataset we used chromosome 1 of Hs. We created five instances S_1, S_2, S_3, S_4, and S_5 of this sequence by randomly choosing 10%, 20%, 30%, 40%, and 50% of the positions, respectively, and randomly replacing the corresponding letters to one of the four letters of the DNA alphabet. We use the uniform distribution for random selections. We computed all minimal absent words for each instance. We considered both the $5' \to 3'$ and the $3' \to 5'$ DNA strands. Table 8.8 depicts elapsed-time comparisons of MAW and PFG. MAW is the fastest in all cases.

Sequence	Size (bp)	MAW (s)	PFG (s)
S_1	248,956,422	435.63	746.93
S_2	248,956,422	438.52	733.69
S_3	248,956,422	444.62	726.34
S_4	248,956,422	444.06	743.29
S_5	248,956,422	449.25	741.01

Table 8.8: Elapsed-time comparison of MAW and PFG for computing all minimal absent words in synthetic data.

8.3 pMAW

In this section, we present algorithm pMAW [99], a new $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm for computing all minimal absent words of a word of length n using arrays SA and LCP. We present our algorithm in detail; then, we show how it can be adapted for parallel computing, achieving near-optimal speed-ups when excluding the indexing data-structure construction time. Our first algorithm MAW is quite straightforward, but there is some redundancy. For example in Table 8.3, we can see that rows 0 and 7, 10 and 13, are identical. Here we try to improve this by considering each right maximal repeated pair (see Definition 6.2.2) only once. To do so, we remark that right maximal repeated pairs are related to the notion of LCP-interval (see Definition 7.4.2). Indeed if the deepest LCP-interval including positions i and j is of depth d then $(i, j, S[\mathsf{SA}[i] \,..\, \mathsf{SA}[i] + d - 1])$ is a right maximal repeated pair. Reciprocaly if (i, j, w) is a right maximal repeated pair then i and j are in the same LCP-interval of depth $|w|$. Thus the computation of the minimal absent words inside a LCP-interval is independent of the computation of minimal absent word from LCP-intervals that are not overlapping.

8.3.1 Computation of Minimal Absent Words

For now on we denote minimal absent words by MAWs. If i is a local maximum in the LCP array, then $[i-1, i]$ is the LCP-interval of LCP-depth $\mathsf{LCP}[i]$ that contains i. Thus we can compute MAWs inside this interval independently of the rest of the array. Our idea is to start the computation at the first local maximum of the LCP array and to visit the surrounding positions in decreasing order of their LCP value. In this process we keep in the array SetLetter the set of letters that occur before the repeated factor. When we reach a local minimum we store its position on the SA array in the stack LifoPos, and the current array SetLetter in the stack LifoSet. We will analyse them once we have visited their whole LCP-interval. This way, we consider each maximal repeated pair and infer from them the whole set of MAWs using Lemma 6.2.1. An example of this function is illustrated in Fig. 8.2.

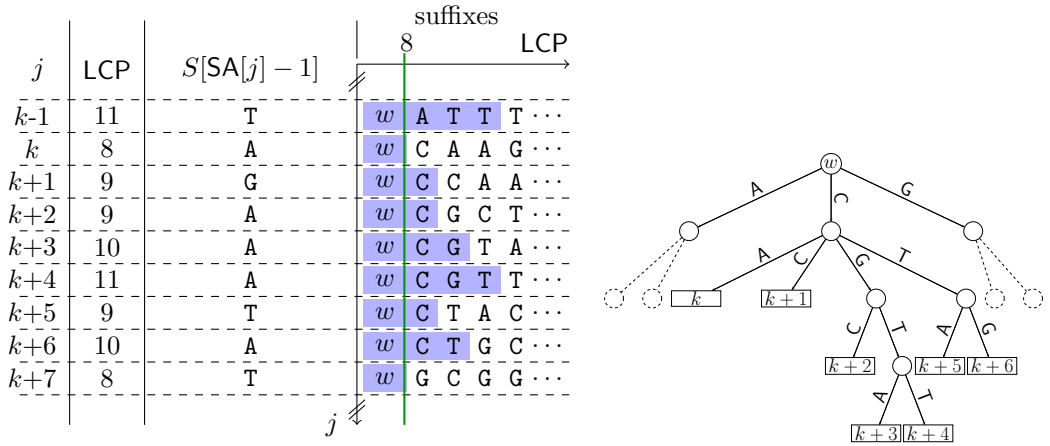
We first pre-compute SA, LCP, and a bit vector v such that $v[i] = 1$ if and only if $\mathsf{LCP}[i]$ is a local maximum. We use rank and select data structures and denote by $rank(k)$ the operation giving the number of '1's in $[0 : k)$ and by $select(k)$ the operation giving the position of the k^{th} 1. The following function presents MAWs computation inside a given interval $[k_1, k_2)$ of SA and LCP.

Function *ComputeMaws* (k_1, k_2, S, SA, LCP, *rank*, *select*)
  SetLetter←$\emptyset$; LifoPos.push(0); LifoSet.push(SetLetter);
  foreach $t \in [rank(k_1)+1 : rank(k_2)]$ **do**
    i←*select*(t); *left*←$i-1$; *right*←$i+1$;
    pos←LifoPos.top(); *lpos*←LCP[*pos*]; SetLetter←$\emptyset$;
    while *1* **do**
      while *pos* > 0 and LCP[i] $<$ *lpos* **do**
        we pop from LifoPos the positions with a LCP value equal to *lpos*; we pop their set of letters from LifoSet;
        we have visited the whole LCP-interval of depth *lpos*, so we infer MAWs using these sets and SetLetter;
        we update *left* and *right*;
        pos←LifoPos.top(); *lpos*←LCP[*pos*];
      if LCP[i] $>$ max(LCP[*left*], LCP[*right*], *lpos*) **then**
        we have visited the whole LCP-interval of depth LCP[i], so we infer MAWs with SetLetter, S[SA[i]-1], and S[SA[*left*]-1];
      SetLetter←SetLetter $\cup \{S[\mathsf{SA}[i]-1]\}$;
      if LCP[*left*] $=$ LCP[i] *or* LCP[*right*] $=$ LCP[i] **then**
        LifoPos.push(i); LifoSet.push(SetLetter);
        we push onto LifoPos all the successive neighbours of interval (*left*,*right*) with a LCP value equal to LCP[i];
        for each of them we push onto LifoSet the letter preceding their corresponding suffix;
        we update *left* and *right*;
      if LCP[*right*] $\leq$ LCP[*left*] $<$ LCP[i] **then** i←*left*; *left*←$i-1$;
      else if LCP[*right*] $>$ LCP[i] **then** we push onto stacks the positions skipped and their corresponding set of letters; **break**;
      else i←*right*; *right*←$i+1$;

Contrary to MAW [130], the previous linear-time algorithm, in pMAW we do not consider our data structures globally; we rather consider each LCP-interval *independently*. This important property will allow us to use parallel computation, as shown in the following paragraph.

Overall Complexity. We use arrays SA and LCP, which can be computed in time and space $\mathcal{O}(n)$ [115]. There also exists a representation which uses $n + o(n)$ bits of storage space and supports rank and select on a bit-vector of size n in constant time [131]. We also use two stacks, LifoPos and LifoSet, where we push and pop $\mathcal{O}(n)$ elements, each containing at most σ integers. Thus the whole algorithm requires time and space $\mathcal{O}(\sigma n)$. We obtain the following result.

Theorem 8.3.1. *Algorithm pMAW solves problem MAW in time and space* $\mathcal{O}(n)$.



(a) **(b)**

step	i	*left*	*right*	SetLetter	Inferred MAWs and action on stacks
1	k+4	k+3	k+5	$\emptyset$	
2	k+3	k+2	k+5	{A}	
3	k+2	k+1	k+5	{A}	we push k+2, k+1, and k+5 onto LifoPos; we push SetLetter,
		k	k+6		{G}, and {T} onto LifoSet
4	k+6	k+5	k+7	$\emptyset$	we infer 2 MAWs: AwCTA, TwCTG
5	k+5	k+4	k+7	{A}	k+5 is already in LifoPos
6	k+7	k+4	k+8	{A,T}	we pop k+5, k+1, and k+2 from LifoPos and {T}, {G}, {A} from LifoSet
		k		{A,G,T}	we infer 7 MAWs: GwCA, TwCA, AwCC, TwCC, GwCG, TwCG, GwCT

(c)

Figure 8.2: Illustration of the algorithm step by step for the interval $[k, k+7)$, $w =$ TCTGAGCG is the common prefix of the considered suffixes.The example is taken from the *Lactobacillus casei* genome (Accession #: NC010999) and $k = 2,554,910$. a) The portions of the LCP array and the letters preceding the suffixes needed for computation. We also show the suffixes, in blue are their prefix of size LCP$[j]$; b) The subtree of the suffix tree corresponding to the interval $[k, k+7)$, it is only an illustration, we do not need it for the computation; c) The update of the different variables and stacks step by step and the computation of minimal absent words

8.3.2 Parallelisation Scheme

We observe that the length of the shortest minimal absent word is equal to the length of the shortest absent word. Now, we show how we can divide the computation into independent tasks.

Lemma 8.3.2. *Let S be a sequence of length n over an alphabet of size σ and let ℓ be the length of the shortest minimal absent word of S. Then the following hold:*

- *For all $k \in [0, \ell-2]$, $|\{s \in [0, n-1] : \mathsf{LCP}[s] = k\}| = (\sigma-1)\sigma^k + 1$;*
- *For all $k \in [\ell-1, n-1]$, $|\{s \in [0, n-1] : \mathsf{LCP}[s] = k\}| < (\sigma-1)\sigma^k + 1$.*

Proof. Let $k \in [0, n-1]$, we denote by $s_0, \ldots, s_{m-1}$, ordered increasingly, the m elements of the set $\{s \in [0, n-1] : \mathsf{LCP}[s] = k\}$. For all $i \in [0, m-1]$, we have $S[\mathsf{SA}[s_i - 1] \,..\, \mathsf{SA}[s_i - 1] + k - 1] = S[\mathsf{SA}[s_i] \,..\, \mathsf{SA}[s_i] + k - 1]$ and $S[\mathsf{SA}[s_i - 1] + k] < S[\mathsf{SA}[s_i] + k]$. We consider the pair (s_i, s_{i+1}) with $i \in [0, m-2]$, there are two cases:

- $\mathsf{lcp}(s_i, s_{i+1}) = k$, so $S[\mathsf{SA}[s_i] \,..\, \mathsf{SA}[s_i] + k - 1] = S[\mathsf{SA}[s_{i+1}] \,..\, \mathsf{SA}[s_{i+1}] + k - 1]$ and $S[\mathsf{SA}[s_i - 1] + k] < S[\mathsf{SA}[s_i] + k] \leq S[\mathsf{SA}[s_{i+1} - 1] + k] < S[\mathsf{SA}[s_{i+1}] + k]$. The alphabet is of size σ; this can happen at most $\sigma - 2$ times consecutively.

- $\mathsf{lcp}(s_i, s_{i+1}) < k$, so $S[\mathsf{SA}[s_i] \,..\, \mathsf{SA}[s_i] + k - 1] < S[\mathsf{SA}[s_{i+1}] \,..\, \mathsf{SA}[s_{i+1}] + k - 1]$. There are σ^k different words of length k; this can happen at most $\sigma^k - 1$ times.

In the first case, we have an additional sub-case, when $\mathsf{SA}[s_i - 1] + k = n$. Then $S[\mathsf{SA}[s_i - 1] + k]$ is not a letter of the alphabet Σ, so we have one more position with a LCP value equal to k. Thus, there are at most $(\sigma - 1)\sigma^k$ pairs (s_i, s_{i+1}), so there are at most $(\sigma - 1)\sigma^k + 1$ positions with a LCP value equal to k.

The equality holds if and only if all the words of length $k + 1$ appear in S, therefore it holds if and only if $k \in [0, \ell - 2]$. □

By Lemma 8.3.2, the length ℓ of the shortest minimal absent word of some word of length n satisfies: $\ell - 1 = \min\{k \geq 0 : |\{s \in [0, n-1] : \mathsf{LCP}[s] = k\}| < (\sigma - 1)\sigma^k + 1\}$. As the alphabet is of size σ, there are σ^k distinct words of length k, but a sequence S of length n has exactly $n + 1 - k$ factors of length k. Thus, if $\sigma^k > n + 1 - k$ there are absent words of size k in S. Consequently we have $\ell \leq \log_\sigma(n + 1 - \ell) < \log_\sigma(n)$. Thus, we compute ℓ, the length of the shortest minimal absent word, in one pass over the LCP array by counting the number of positions having a LCP value equal to d, for all $d \in [0, \lfloor \log_\sigma(n) \rfloor]$.

According to Lemma 6.2.1 we can ignore positions having a LCP value lower than $\ell - 2$ when computing minimal absent words. Hence, we focus on LCP-intervals of LCP-depth above or equal to $\ell - 2$: they are sufficient to exhaustively compute the set of minimal absent words. Consequently we compute the set of positions k_i with i in $[0, (\sigma - 1)\sigma^{\ell-3}]$ such that $\mathsf{LCP}[k_i] = \ell - 3$. $[0, k_0), [k_0, k_1), \ldots, [k_{m-1}, k_m), [k_m, n)$, with $m = (\sigma - 1)\sigma^{\ell-3}$, is a partition of $[0, n - 1]$. This partition is such that, every LCP-interval of LCP-depth above or equal to $\ell - 2$ is entirely included in one of the sub-intervals $[k_i, k_{i+1})$.

Therefore we can consider each one of these sub-intervals *independently*, and thus parallelise the computation of minimal absent words. In each sub-interval we go through the SA and LCP arrays starting at the first (from left to right) local maximum and going down until we reach a local minimum, as described in Section 8.3.1. For an overview of the algorithm pMAW inspect Fig. 8.3.

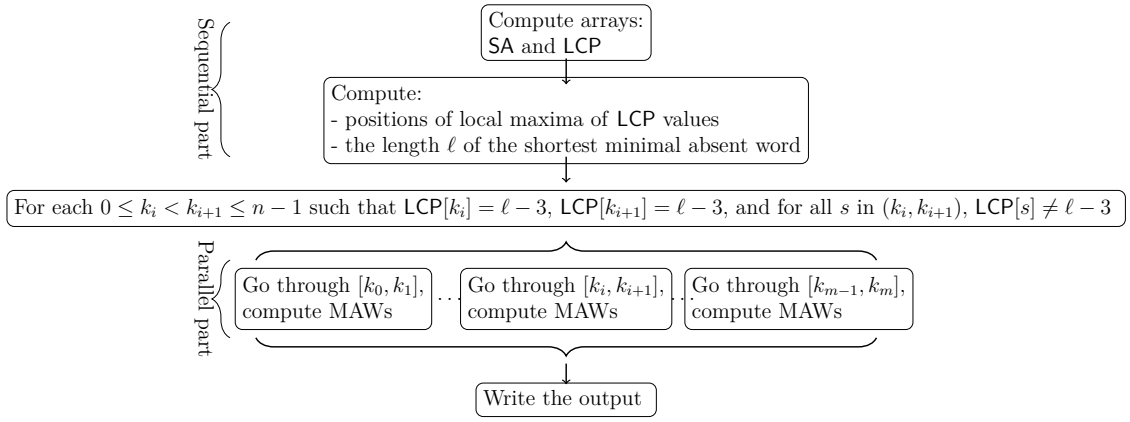
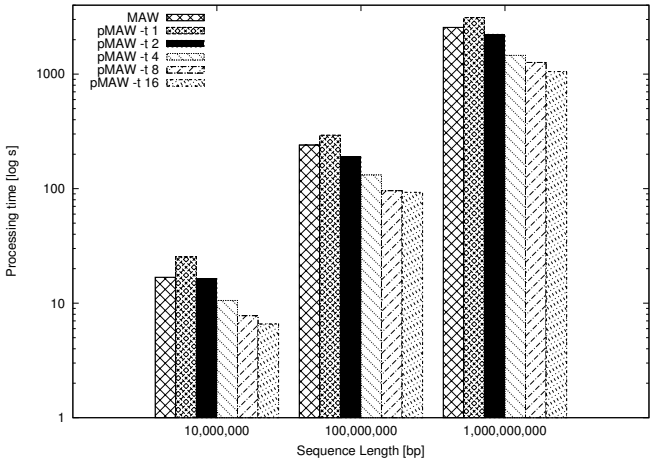


Figure 8.3: Overview of Algorithm pMAW

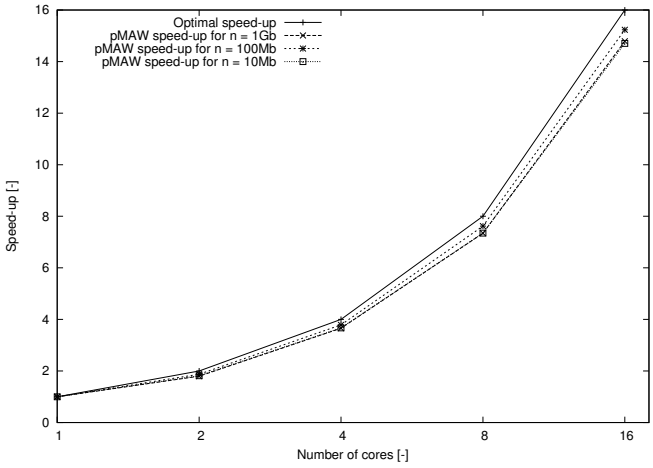
8.3.3 Results

We implemented algorithm pMAW as a program to compute all minimal absent words of a given sequence. The program was implemented in the C programming language, using Open Multi-Processing (OpenMP) API for shared-memory multiprocessing programming, and developed under GNU/Linux operating system. It takes as input arguments a file in (Multi)FASTA format and the minimal and maximal length of minimal absent words to be outputted; and then produces a file with all minimal absent words of length within this range as output. There are additional input parameters; for example, the number t of available processing elements. The implementation is distributed under the GNU General Public License (GPL), and it is available at `http://github.com/solonas13/maw`, which is set up for maintaining the source code and the man-page documentation. The experiments were conducted on a Desktop PC using 1 to 16 cores of 2 Intel Xeon E5-2670V2 Ten-Core CPUs at 2.50GHz and 256GB of main memory under 64-bit GNU/Linux.

To evaluate the efficiency of our implementation, we compared it against the corresponding performance of MAW [130], which was at the time the fastest available implementation for computing minimal absent words. We generated three random sequences of length 10Mbp, 100Mbp, and 1Gbp, respectively, by using a uniform frequency distribution of letters of the DNA alphabet. We computed all minimal absent words of length at most 20 for each sequence. We considered both the $5' \rightarrow 3'$ and the $3' \rightarrow 5'$ DNA strands. Fig. 8.4a depicts elapsed-time comparisons of pMAW and MAW, including the sequential part of the algorithm. pMAW becomes the fastest in all cases when $t \geq 2$ accelerating the computation by more than a factor of two when $t = 16$. Notice that the y-axis is on logarithmic scale. The measured relative speed-up of pMAW is illustrated in Fig. 8.4b. The relative speed-up was calculated as the ratio of the runtime of pMAW on 1 core to the runtime



(a) Elapsed-time comparison

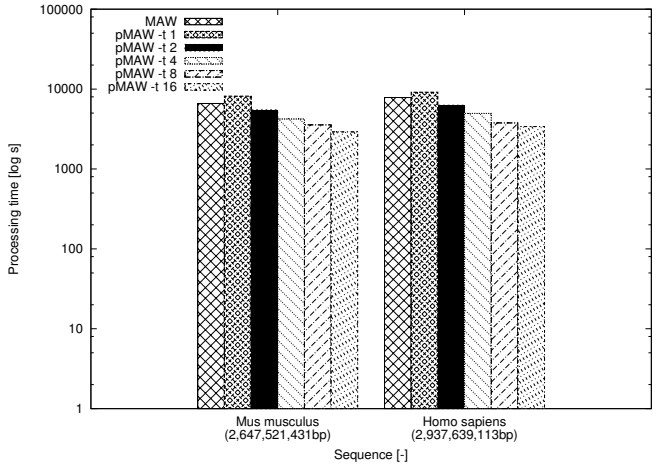


(b) Relative speed-up of pMAW

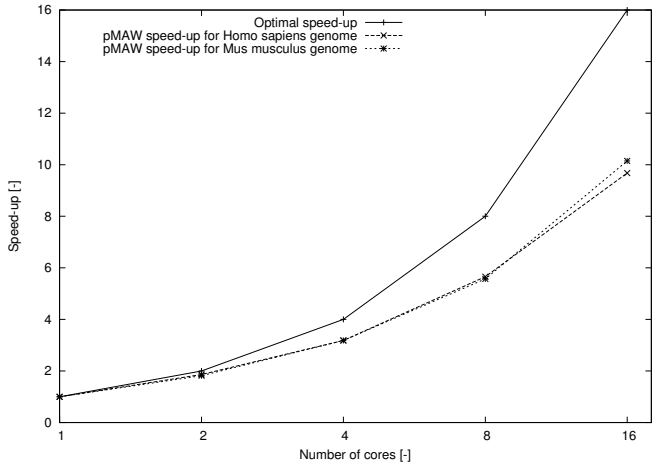
Figure 8.4: Elapsed-time comparison of pMAW and MAW and relative speed-up of pMAW for computing minimal absent words using synthetic DNA sequences

of pMAW on t cores, excluding the sequential part of the algorithm. The results highlight the excellent scalability of pMAW when the letters have a uniform frequency distribution in the sequence. In this case, pMAW achieves near-optimal speed-ups, confirming our theoretical findings.

To further evaluate the efficiency of our implementation, we compared it against the corresponding performance of MAW using real data. We considered the genomes of *Homo sapiens* and *Mus musculus*, obtained from the NCBI database (`ftp://ftp.ncbi.nih.gov/genomes/`). We computed all minimal absent words of length at most 20 of the complete sequence of the *Homo sapiens* $(2,937,639,113\text{bp})$ and *Mus musculus* $(2,647,521,431\text{bp})$ genomes—ignoring unknown bases. We considered both the $5' \to 3'$ and the $3' \to 5'$ DNA strands. Fig. 8.5a depicts elapsed-time comparisons of pMAW and MAW, including the sequential part of the algorithm.



(a) Elapsed-time comparison



(b) Relative speed-up of pMAW

Figure 8.5: Elapsed-time comparison of pMAW and MAW and relative speed-up of pMAW for computing minimal absent words using real DNA sequences

pMAW becomes the fastest in all cases when $t \geq 2$ accelerating the computation by more than a factor of two when $t = 16$. Notice that the y-axis is on logarithmic scale. The measured relative speed-up of pMAW is illustrated in Fig. 8.5b. The relative speed-up was calculated as the ratio of the runtime of pMAW on 1 core to the runtime of pMAW on t cores, excluding the sequential part of the algorithm. The results highlight the good scalability of pMAW with real data. The computation is accelerated by a factor of 10 when $t = 16$. The maximum allocated memory was 137GB for both programs.

The importance of our contribution here is underlined by the fact that any parallel algorithms for the construction of the involved indexing data structure can be used directly to replace the sequential part of the algorithm proposed here (see

Fig. 8.3). This would result in a fully parallel algorithm for the computation of minimal absent words.

8.4 em-MAW

After providing a fast implementation due to a parallel algorithm, we aim to reduce the bottleneck of our previous implementations, the RAM usage. We propose an adaptation of MAW to the external memory (EM) model of computation (see paragraph 7.3.2 and the reference [118] for details). We recall that by M we denote the RAM (internal memory) size and by B the disk (external memory) block size, smaller than M, both measured in units of $\Theta(\log n)$-bit words. We further assume that $M = \Omega(\log n)$ and $M = \mathcal{O}(n)$. In the EM model, each transfer of B words between memory and disk is called an IO, and, hence, an algorithm's complexity is mainly measured in IOs.

Our algorithm for computing minimal absent words in external memory, which we denote by em-MAW has the following three main stages:

8.4.1 Stage 1: Computing SA, LCP, and BWT

Computing SA and LCP. There exist IO-optimal algorithms for computing SA and LCP, but they use large amounts of disk, which can be problematic in practice. In our implementation we instead make use of the space-efficient pSAscan algorithm due to Kärkkäinen et al. [120] to compute SA. The IO complexity of pSAscan is $\mathcal{O}(\frac{n^2}{MB\log_\sigma n} + \frac{n}{B})$ and its time complexity is $\mathcal{O}(\frac{n^2}{M}\log(2 + \frac{\log_\sigma}{\log\log n}))$. To compute LCP we make use of the external-memory LCPscan algorithm, due to Kärkkäinen and Kempa [122]. It leads to a quadratic-time complexity of time $\mathcal{O}(\frac{n^2}{M\log_\sigma n} + n\log_{\frac{M}{B}}\frac{n}{B})$ with $\mathcal{O}(\frac{n^2}{MB(\log_\sigma n)^2} + \frac{n}{B}\log_{\frac{M}{B}}\frac{n}{B})$ IOs.

Computing BWT. There are no IO-optimal algorithms for computing BWT directly from S. In our implementation we use the following easy-to-implement (and non-optimal) method, which takes time $\mathcal{O}(n\log M + \frac{n^2}{M})$ with $\mathcal{O}(\frac{n^2}{MB})$ IOs. If the RAM is not enough for the word to fit inside, we compute the BWT block by block. We store in memory m pairs of the form $(i, \mathsf{SA}[i])$ such that they fit in RAM, and we sort the pairs with respect to the $\mathsf{SA}[i]$ field. Then we scan S and the list of sorted pairs. During the scan, we replace the $\mathsf{SA}[i]$ field of each pair with letter $S[\mathsf{SA}[i]-1]$ (except if $\mathsf{SA}[i] = 0$, in which case we replace it with a letter '#' not from Σ). Finally, we sort the pairs with respect to the i field. The letters are a contiguous segment of the BWT, we store them. We repeat the process, until we have the whole BWT.

8.4.2 Stage 2: Computing sets $B_1[j]$ and $B_2[j]$

Given the input sequence S and its SA and LCP in internal memory, the computation of sets $B_1[j]$ and $B_2[j]$ can be done in internal memory in time and space $\mathcal{O}(n)$ as shown in Lemma 8.2.4. In our algorithm, we adapt the algorithm MAW, to compute sets $B_1[j]$ and $B_2[j]$ in external memory, when we have SA, LCP, and BWT precomputed and stored in external memory. The main difference is that we do not use the sequence S itself but rather its BWT. To compute the sets $B_1[j]$ and $B_2[j]$, we scan SA, LCP, and BWT twice: top-down and bottom-up. These data structures are always accessed sequentially. Thus we can store them in external memory and then scan or modify them by transferring in RAM only a segment of entries, whose number is proportional to M. Transferring an array of size n from or to external memory requires time $\mathcal{O}(n)$ with $\mathcal{O}(\frac{n}{B})$ IOs [118].

8.4.3 Stage 3: Computing the set of minimal absent words

At this point we have stored into external memory the sets $B_1[j]$ and $B_2[j]$ for all $j \in [0 : 2n-1]$. By applying Lemma 8.2.4 we can obtain all minimal absent words of S by computing the difference $B_2[j] \setminus B_1[j]$ for all $j \in [0 : 2n-1]$. Hence we obtain the following.

Theorem 8.4.1. *Given a word of length n and its SA, LCP, and BWT in external memory, algorithm emMAW computes all minimal absent words in time $\mathcal{O}(n)$, with $\mathcal{O}(\frac{n}{B})$ IOs, and using $\mathcal{O}(n)$ space in external memory.*

Note that asymptotically, the size M of the RAM, does not have an influence on the number of IOs. However, the computation time decreases as the size of RAM increases, until it reaches a lower bound. This lower bound corresponds to the size of RAM that is large enough to store all stacks.

8.4.4 Results

We implemented algorithm emMAW as a program to compute all minimal absent words of a given sequence. The program was implemented in the C programming language. The implementation is available at `http://github.com/solonas13/maw` under the GNU GPL terms. We made use of the following two machines in order to evaluate our implementation. The first one, denoted by **M1**, is a Desktop PC with 8 cores of 1 Intel(R) Core(TM) i7-4790 CPU at 3.60GHz with 8M Cache and 16GB of DDR3 RAM under 64-bit GNU/Linux. **M1** was equipped with a single Samsung SSD 850 PRO Series disk with capacity 256GB. The second one,

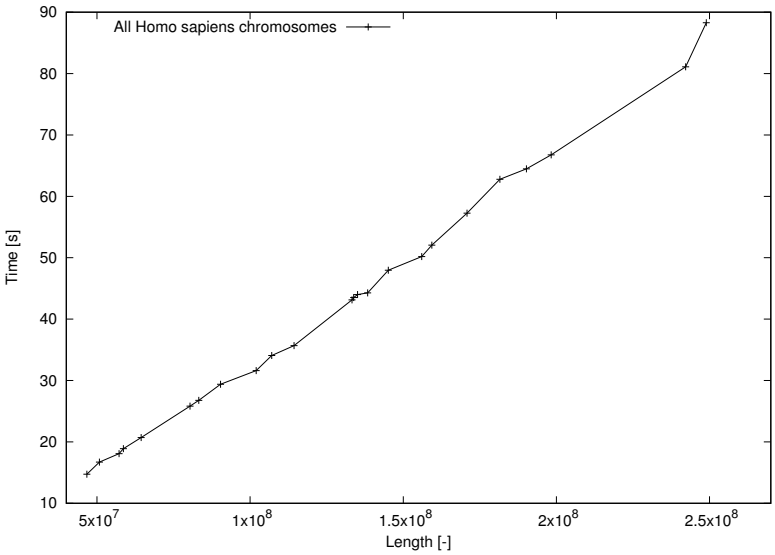
denoted by **M2**, is a single node of a cluster computer with 2×10 cores of Intel(R) Xeon(R) CPU E5-2660 v3 at 2.60GHz with 25M Cache and 384GB of DDR3 RAM under 64-bit GNU/Linux. **M2** was equipped with a single Samsung SSD 850 PRO Series disk with capacity 256GB.

External memory

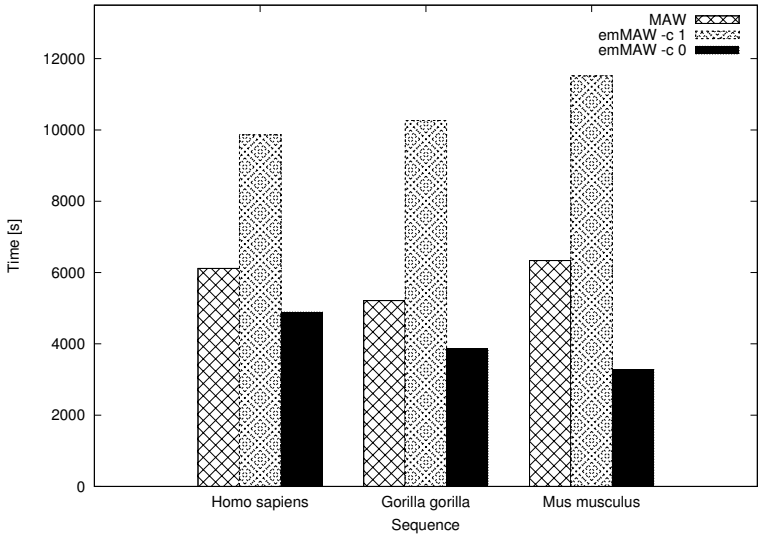
Our first task was to validate our theoretical findings (Theorem 8.4.1). To this end, we used as input all chromosome sequences of the *Homo sapiens* genome obtained from the NCBI database (`ftp://ftp.ncbi.nih.gov/genomes/`). We computed all minimal absent words of length at most 11 for each sequence separately. We considered only the $5' \rightarrow 3'$ DNA strand. We had first pre-computed and stored in external memory the necessary data structures. This set of runs was conducted on **M1**. Fig. 8.6(a) depicts elapsed-time measurements of emMAW (without accounting for the time to construct the data structures) using only 500 MiB of internal memory. The results confirm our theoretical findings: the elapsed-time increases linearly with the length of the input sequence. To further evaluate the efficiency of our implementation, we used as input the full genome of *Homo sapiens* without pre-computing the necessary data structures. We computed all minimal absent words of length at most 11 using only 1,000 MiB of internal memory. We considered both DNA strands. The whole assignment, the construction of the data structures plus the computation of minimal absent words, took less than 4 hours to finish.

Internal memory

We next compared the efficiency of emMAW against the corresponding one of MAW [130], the fastest internal-memory implementation, when both exclusively use internal memory (150,000 MiB) for their computations. We considered the full genomes of *Homo sapiens*, *Gorilla gorilla*, and *Mus musculus* genomes, obtained from the NCBI database. We computed all minimal absent words of length at most 11 for each sequence. We considered both DNA strands. For this set of runs, we used **M2** to ensure that the necessary data structures can be constructed and stored in internal memory. We used two options for emMAW: (i) `-c 1` denoting that the necessary data structures must be constructed; (ii) `-c 0` denoting that they have already been pre-computed and can be read from disk. Elapsed-time comparison is illustrated in Fig. 8.6(b). The results show that emMAW is less than two times slower than MAW with `-c 1`; most importantly, we see that emMAW becomes faster than MAW with `-c 0`.



(a) Elapsed-time of emMAW in external memory



(b) Elapsed-time comparison of MAW and emMAW in internal memory

Figure 8.6: Computing minimal absent words in internal and external memory

8.4.5 Conclusion

We presented algorithm emMAW, the first external-memory algorithm for computing minimal absent words. Given a sequence of length n and its SA, LCP, and BWT in external memory, emMAW computes all minimal absent words in time $\mathcal{O}(n)$, with $\mathcal{O}(\frac{n}{B})$ IOs, and using $\mathcal{O}(n)$ space in external memory. We also made available an open-source implementation of emMAW, `http://github.com/solonas13/maw`. We provided experimental results showing that our implementation requires less than 4 hours on a standard workstation to process the full human genome when

as little as 1 GB of RAM is made available. Our implementation, despite making use of external memory, is fast; indeed, even on relatively smaller data sets when enough RAM is available to hold all necessary data structures, it is less than two times slower than state-of-the-art internal-memory implementations.

9

Minimal absent words in a sliding window

Contents

9.1 Motivations

We have seen several linear time and space algorithms to compute minimal absent words. The one using external memory gives very interesting results as it runs on the whole human genome on a desktop computer in a few hours. However, even though the RAM size can be as small as the user desires (up to a few kB depending on the size of the alphabet), the disk usage remains a bottleneck if we want to consider very huge sequences. Indeed for the computation we need to store several tables of size linear in the size of the input sequence.

The problem we want to tackle is to compute the minimal absent words in a sliding window. This way, we can analyse a huge sequence of size n by focusing our

attention onto all its factors of a fixed size m. For example, given a pattern P we can find the most similar region of the sequence y by using similarity measure based on minimal absent words. An other application to this problem is the data compression, we can thus compress in an on-line manner an input stream, by compressing it chunk by chunk using a minimal absent words compression methods [94].

Problem 2: Computation of Minimal Absent Words in a sliding window
Input: A sequence y of length n on a fixed size alphabet Σ, the size m of the window of interest.
Output: Maintain the set of minimal absent words of a window of size m sliding along the sequence y.

9.2 On-line computation of minimal absent words

Ota et al. [132] were the first to propose an on-line algorithm to compute the set of minimal absent words (also called *antidictionnary*). Their construction algorithm is based on two algorithms for building suffix trees, the Ukkonen algorithm [103] and the Weiner algorithm [101]. They proved that the total construction complexity is linear, but we suspect that it might be slow in practice. Indeed during the construction algorithm they maintain two suffix trees, the suffix tree of the input sequence using Ukkonen algorithm, and the suffix tree of the reverse sequence using Weiner algorithm. Consequently their algorithm is not space efficient as these two structures are redundant. However, we used their approach and combined it with a construction algorithm of the suffix tree for a sliding window.

We present the first algorithm that can solve Problem 2 in linear time on the size of the input sequence and in linear space on the size of the sliding window.

9.3 Computation of the suffix tree for a sliding window

The algorithm of Ukkonen (see a brief explanation in paragraph 7.2.3) constructs the suffix tree on-line in $\mathcal{O}(n)$ time for a constant-sized alphabet by processing the input word from left to right. To adapt it for a sliding window with amortized constant time per one window shift, two additional problems need to be resolved: (i) deleting the leftmost letter of a window; and (ii) maintaining edge labels under window shifts. We briefly explain how to achieve this, more explanations can be found in the original paper [133].

Removing the leftmost letter. Consider the longest repeated prefix of the current window. When the leftmost letter is deleted, all prefixes that are longer than this prefix need to be removed from the tree but the longest repeated prefix and all shorter prefixes will remain in the tree. To remove these prefixes we delete the leaf corresponding to the whole window and its incoming edge as follows:

- If the longest repeated prefix corresponds to an explicit node, this node is the parent of the leaf to be deleted. If this node has only one child remaining, we delete the node and merge the two edges. Otherwise, we do nothing.

- If the longest repeated prefix corresponds to an implicit node, it is equal to the longest repeated suffix. We create a new leaf in place of the one we have deleted. We label it with the starting position of what was the longest repeated suffix and its incoming edge is labelled accordingly. Finally, we add a suffix link from the last created leaf to this leaf and move up the active node by following the suffix link.

Maintaining Edge Labels. Assume by induction that all edge labels are correctly positioned relative to the current window. For the next m shifts of the window, we still maintain the same relative positioning of edge labels. After the m shifts, edge labels are recomputed by a bottom-up traversal of the tree. Since m shifts create at most $2m$ nodes, the amortized time spent on one shift is $O(1)$.

9.4 Combinatorial results

In this section we consider a word z of fixed length m on an alphabet Σ of size σ and denote by $M(z)$ its set of MAWs. The word z essentially represents the content of the window on word y used in the algorithm of Section 9.5. We first discuss changes to be done on the set of MAWs when appending and removing letters on the word of interest. Then we show bounds on the number of changes on the set of MAWs when moving forward the current window by one position.

9.4.1 Changes when appending one letter to the window

We denote by $M(z)|\alpha$, $\alpha \in \Sigma$, the operation on the set of MAWs when concatenating the letter α to the, possibly empty, word z. The operation creates $M(z\alpha)$ from $M(z)$. We introduce some bounds on the number of insertions/deletions for the on-line computation of the set of MAWs. These results have already been shown in [132] and we briefly present them for completeness.

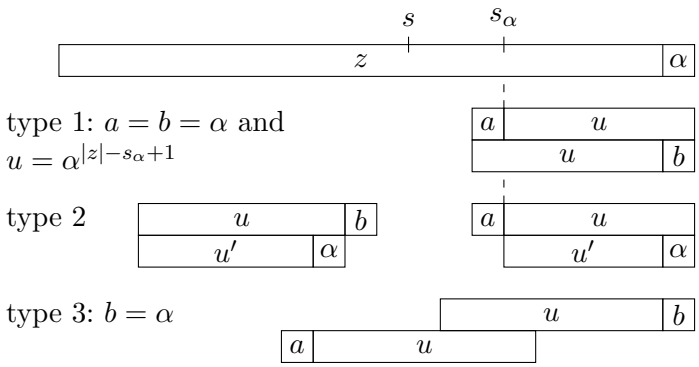


Figure 9.1: Illustration of the three different types of MAWs that are added when letter α is appended to z.

We denote by s the starting position of the longest suffix of z that repeats in z; when this suffix is empty we set $s = |z|$. We also denote by s_α the starting position of the longest suffix that occurs in z followed by α; when this suffix is empty we set $s_\alpha = |z|$. Note that we have $s \leq s_\alpha$ because the latter suffix obviously repeats in z. This is illustrated in Figure 9.1.

The next two lemmas state bounds of the number of insert and delete operations performed by $M(z)|\alpha$.

Lemma 9.4.1. *$M(z)|\alpha$ deletes exactly one MAW from $M(z)$, namely $z[s_\alpha-1\,..\,|z|-1]\alpha$*

Proof. Let $w = aub$, $a, b \in \Sigma$ and $u \in \Sigma^*$, be a MAW to be removed. This means that aub is absent in z but present in $z\alpha$. Thus $b = \alpha$ and au is a suffix of z that does not occur followed by α in z. The word $ub = u\alpha$ is also present in z, so u is a suffix of z that occurs in z followed by α. Then the starting position of the suffix occurrence u in z is s_α and $w = z[s_\alpha - 1\,..\,|z| - 1]\alpha$. □

To establish an upper bound on the number of MAWs added by the operation $M(z)|\alpha$, we first divide the new MAWs of the form aub, $a, b \in \Sigma$ and $u \in \Sigma^*$, into three types (see also Figure 9.1):

1. au and ub are absent in z.

2. au is absent in z and ub is present in z.

3. au is present in z and ub is absent in z.

Lemma 9.4.2. *There are at most one MAW of type 1, σ MAWs of type 2, and $(s_\alpha - s)(\sigma - 1)$ MAWs of type 3, added by the operation $M(z)|\alpha$.*

Proof. We consider a new MAW $w = aub$, $a, b \in \Sigma$ and $u \in \Sigma^*$, created by the operation.

Let w be of type 1, that is, au and ub do not occur in z. Then they are both suffixes of $z\alpha$, and because they have same length, are equal. This implies that u is both a prefix and a suffix of $ub = u\alpha$. Thus the latter has period 1, w is of the form $\alpha^{|w|}$, and $u = \alpha^{|w|-2}$. But then $u\alpha$ is absent in z. Therefore, $\alpha^{|w|-3}$ is the longest repeated suffix of z that occurs followed by α in z. Consequently $|w| = |z| - s_\alpha + 3$.

Let w be of type 2, that is, ub occurs in z and au occurs in $z\alpha$ but not in z. Then au is a suffix of $z\alpha$ and u can be written $u'\alpha$. As ub occurs in z, u' is a suffix of z that occurs in z followed by α. Moreover, since $au = au'\alpha$ does not occur in z, u' is the longest suffix of z that occurs in z followed by α, therefore its starting position as a suffix is s_α. The letter b can be any letter of the alphabet of z that occurs after an occurrence of u in z. Consequently there are at most σ such MAWs.

Let w be of type 3, that is, au occurs in z and ub occurs in $z\alpha$ but not in z. This implies that $b = \alpha$, u is a suffix of z not preceded by a, and au occurs elsewhere in z. Since no occurrence of u in z is followed by α, we have that the starting position k of u as a suffix satisfies $s \leq k < s_\alpha$. Therefore, there are at most $s_\alpha - s$ possible words u and for each of them, there are at most $\sigma - 1$ possibilities for the letter a to obtain a MAW. Consequently, there are at most $(s_\alpha - s)(\sigma - 1)$ such MAWs. □

The previous lemma shows that during one step of the computation of MAWs for a sliding window we may have to handle $\mathcal{O}(\sigma m)$ new MAWs. Indeed the example of $M(\mathsf{AC}^{m-1})|\mathsf{A}$ shows that the bound is tight. However, the total number of insertions when computing the set of MAWs for a word y of length n get amortized to $\mathcal{O}(\sigma n)$ in an on-line computation.

Proposition 9.4.3 ([132])**.** *Starting with the empty word, and applying n times the operation $|$ leads to a total number of insertions/deletions of MAWs in $\mathcal{O}(\sigma n)$.*

Proof. The number of MAWs of the whole word of length n is in $\mathcal{O}(\sigma n)$. As stated by Lemma 9.4.1 at most one MAW can be deleted by each application of the operation $|$. Thus the total number of insertions/deletions is still in $\mathcal{O}(\sigma n)$. □

9.4.2 Changes when removing the first letter of the window

Removing the leftmost letter of the window is a dual question to what is done previously. We now focus on the longest repeated prefix instead of the longest repeated suffix.

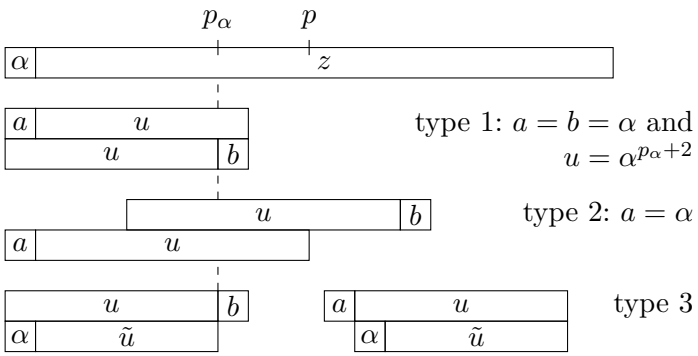


Figure 9.2: Illustration of the three different types of MAWs that are deleted when removing α, the letter before z.

Let us denote by p the ending position of the longest repeated prefix of z and by p_α the ending position of the longest prefix of z that occurs in z preceded by α. We set them to 0 when the prefixes are empty. Note that $p_\alpha \leq p$. Similarly as Lemma 9.4.1, removing a letter from the left creates exactly one MAW.

Lemma 9.4.4. *The operation $M(\alpha z) \to M(z)$ creates exactly one MAW, which is $\alpha z[0\,..\,p_\alpha + 1]$.*

Similarly as in Section 9.4.1, we distinguish three types of MAWs to be deleted by the operation:

1. au and ub are absent in z.

2. au is absent in z and ub present in z.

3. ub is absent in z and au present in z.

We note that types 1, 2 and 3 behave respectively similarly to type 1, 3 and 2 in Section 9.4.1; see Figure 9.2 for an illustration. The following result is similar to that stated in Lemma 9.4.2.

Lemma 9.4.5. *There are at most one MAW of type 1, $(\sigma - 1)(p - p_\alpha)$ MAWs of type 2, and σ MAWs of type 3, to be deleted by the operation $M(\alpha y) \to M(y)$.*

9.4.3 Changes when sliding a window over a text

We now focus on our main problem: MAWs in a sliding window. For $m < n$ and for all i, $0 \leq i \leq n - m$, we consider the window $y[i\,..\,i + m - 1]$ and define:

- s_i the starting position of its longest repeated suffix,

- $\tilde{s_i}$ the starting position of its longest suffix that occurs followed by $y[i + m]$,

- ss_i the starting position of its longest suffix that is a power,
- p_i the ending position of its longest repeated prefix,
- $\tilde{p_i}$ the ending position of its longest prefix that occurs preceded by $y[i-1]$,
- pp_i the ending position of its longest prefix that is a power.

In what follows, we make use of this notation considering the case of a sliding window. The following lemma shows that we cannot output in linear time the set of MAWs in the sliding window at each step of the process.

Lemma 9.4.6. *The upper bound of* $\sum_{i=0}^{n-m} |M(y[i\,..\,i+m-1])|$ *is* $\mathcal{O}(\sigma nm)$ *and this bound is tight.*

Proof. For every factor z of length m of y, $|M(z)|$ is $\mathcal{O}(\sigma m)$. Thus the upper bound of their sum is $\mathcal{O}(\sigma nm)$. Now consider $y = (\mathsf{A}^{m-1}\mathsf{C}^{m-1})^{\frac{n}{2m-2}}$ of length n and its factors of length $2m$. In each factor w of length $2m$, this kind of pattern occurs: $XY^{m-1}X$, with $\{X, Y\} = \{\mathsf{A}, \mathsf{C}\}$. Thus $\{XY^iX | 1 \leq i \leq m-1\} \subseteq M(w)$, so $|M(w)| \geq m-1$. Consequently the bound is tight. One can generalize this construction of y to obtain a tight bound for larger alphabets (Lemma 1 in [134]). □

However, as shown below, we can bound the number of changes necessary to maintain the set of MAWs for a sliding window. We obtain the following result.

Theorem 9.4.7. *The upper bound of* $\sum_{i=0}^{n-m-1} |M(y[i\,..\,i+m-1]) \triangle M(y[i+1\,..\,i+m])|$ *is in* $\mathcal{O}(\sigma n)$.

Proof. Let us consider the set $M(y[i\,..\,i+m-1]) \triangle M(y[i\,..\,i+m])$ with $0 \leq i < n-m$. From Lemmas 9.4.2 and 9.4.1 we get

$$|M(y[i\,..\,i+m-1]) \triangle M(y[i\,..\,i+m])| \leq (\tilde{s_i} - s_i)(\sigma - 1) + \sigma + 2.$$

Then,

$$\sum_{i=0}^{n-m-1} |M(y[i\,..\,i+m-1]) \triangle M(y[i\,..\,i+m])| \leq \sum_{i=0}^{n-m-1} (\tilde{s_i} - s_i)(\sigma - 1) + n\sigma + 2n.$$

We note that $\tilde{s_i} \leq s_{i+1} \leq \tilde{s_i} + 1$ and we have $s_i \leq \tilde{s_i}$ thus

$$0 \leq \sum_{i=0}^{n-m-1} (\tilde{s_i} - s_i) = \sum_{i=0}^{n-m-1} \tilde{s_i} - \sum_{i=0}^{n-m-1} s_i$$

$$0 \leq \sum_{i=0}^{n-m-1} (\tilde{s_i} - s_i) = \tilde{s}_{n-m-1} - s_0 + \sum_{i=0}^{n-m-2} (\tilde{s_i} - s_{i+1}) \leq n$$

Then $\sum_{i=0}^{n-m-1} |M(y[i\,..\,i+m-1]) \triangle M(y[i\,..\,i+m])| \leq 2n\sigma + n$. Now, we consider the set $M(y[i\,..\,i+m]) \triangle M(y[i+1\,..\,i+m])$, with Lemmas 9.4.4 and 9.4.5 we obtain a similar inequality: $\sum_{i=0}^{n-m-1} |M(y[i\,..\,i+m]) \triangle M(y[i+1\,..\,i+m])| \leq 2n\sigma + n$. Thus we obtain the desired bound by the triangle inequality. □

9.5 Algorithm to compute minimal absent words in a sliding window

Consider a word y of length n on an alphabet Σ of size σ. Our goal is to maintain the set of MAWs for a sliding window of size m. That is, for all successive $i \in [0, \ldots, n-m]$, we want to compute $M_m(i) = M(y[i \,..\, i+m-1])$.

For a word z, by $\Sigma(z)$ we denote the alphabet of z and by $V(z)$ the set of explicit nodes in the suffix tree of z. Consider the map $f : M(z) \to \Sigma(z) \times V(z)$ defined by $f(aub) = (a, v_{ub})$, where $a \in \Sigma$ and v_{ub} is either the explicit node corresponding to the factor ub or the immediate explicit descendant node if this node is implicit.

Lemma 9.5.1. *Map f is an injection.*

Proof. Let $w, w' \in M(z)$, $w \neq w'$, $w = aub$ and $w' = a'u'b'$, with $a, b, a', b' \in \Sigma(z)$ and $u, u' \in \Sigma(z)^*$.

Suppose that $f(w) = f(w')$, then $a = a'$ and $v_{ub} = v_{u'b'}$. Thus ub and $u'b'$ are distinct prefixes of the factor corresponding to v_{ub}, consequently one is prefix of the other, without loss of generality ub is prefix of $u'b'$. Then aub is a prefix of $au'b'$, this is impossible as they are both MAWs of z. Thus two distinct elements of $M(z)$ cannot share the same image by f, so f is an injection. □

Lemma 9.5.1 allows us to represent all MAWs by storing a set of letters in each explicit node of the tree. We will call this set the *maw*-set. Moreover, a letter a in the *maw*-set will be *tagged* iff u corresponds to an implicit node in the tree. Observe that a can become tagged only when u is a repeated suffix of y. This is because factors au and ub define distinct occurrences of u, and the occurrence of au must be a suffix, otherwise u would be followed by two distinct letters and would then be an explicit node. Besides *maw*-sets, we will also need to store in each explicit node another set of letters: the set of all letters preceding the occurrences of the factor corresponding to the node.

By induction, assume we are at position i, the suffix tree $T_m(i)$ is built and the set of MAWs $M_m(i)$ has been computed. We now explain how to update $T_m(i)$ and $M_m(i)$ to obtain $T_m(i+1)$ and $M_m(i+1)$. The tree is updated based on Senft's algorithm, by first adding a letter to the right of the current window and then deleting the leftmost letter. The set of MAWs is updated using Lemmas 9.4.1, 9.4.2 and 9.4.4, 9.4.5 respectively. The algorithm will maintain positions $s_i, p_i, \tilde{s}_i, \tilde{p}_i, ss_i, pp_i$ as defined in Section 9.4.3. We store the leaf nodes in a list so that the last created leaf and the "oldest" leaf currently in the tree can be accessed in constant time.

Adding a letter to the right.

We follow Ukkonen's algorithm for updating the suffix tree. Recall that Ukkonen's algorithm proceeds by updating the *active node* in the tree. At the beginning of each iteration, the active node corresponds to the longest repeated suffix, i.e. to factor $y[s_i \,..\, i+m-1]$. The node corresponding to the longest repeated prefix is called the *head node*.

The algorithm starts from the active node and updates it following the suffix links until reaching a node with an outgoing edge starting with $y[i+m]$ – this node corresponds to the suffix starting at $\tilde{s}_i$. At the same time, we compute MAWs of type 3 that are created. For each $s_i \leq j < \tilde{s}_i$, we perform the following.

- If the active node is implicit we make it explicit. We set its set of preceding letters equal to its child's set. We move the untagged letters of the *maw*-set of its child to the *maw*-set of the active node. We untag the tagged letters of the *maw*-set of its child. If the last node created at this window shift does not have a suffix link, we add a suffix link from this node to the active node. We add the letter corresponding to this suffix link to the set of preceding letters of the active node.

- We create a leaf labelled j, with $y[j-1]$ in its set of preceding letters. We create an edge from the active node to this leaf with the label $y[i+m]$.

- For each letter $a \neq y[j-1]$ in the set of preceding letters of the active node, $ay[s_i + j \,..\, i+m] \in M_{m+1}(i) \backslash M_m(i)$ (type 3 in Lemma 9.4.2), therefore we add a in the *maw*-set of the leaf.

The current active node corresponds to the factor $y[\tilde{s}_i \,..\, i+m-1]$. According to Lemma 9.4.1, there is exactly one MAW to be deleted which is $y[\tilde{s}_i - 1 \,..\, i+m]$. This MAW is stored in the child of the active node by following the edge starting with $y[i+m]$; we remove $y[\tilde{s}_i - 1]$ (tagged or not) from its *maw*-set.

Then we update the active node by following the edge starting with $y[i+m]$; now it corresponds to the factor $y[\tilde{s}_i \,..\, i+m]$. If the head node was also corresponding to the factor $y[\tilde{s}_i \,..\, i+m-1]$, we move it down with the active node; we have $\tilde{p}_{i+1} = p_i + 1$, otherwise we have $\tilde{p}_{i+1} = p_i$. If the active node is explicit, we update its set of preceding letters by adding $y[\tilde{s}_i - 1]$.

Then, for each letter b occurring after an occurrence of $y[\tilde{s}_i \,..\, i+m]$ in $y[i \,..\, i+m-1]$, $y[\tilde{s}_i - 1 \,..\, i+m]b \in M_{m+1}(i) \backslash M_m(i)$ (type 2 in Lemma 9.4.2). These MAWs are stored in their corresponding child of the active node. If the active node is implicit, there is only one of them and we tag the letter.

By Lemma 9.4.2, if $ss_i = \tilde{s}_i - 1$, then $y[i+m]y[\tilde{s}_i - 1\,..\,i+m]$ is the new MAW of type 1. We store it in the *maw*-set of the child of the active node by following the edge starting with $y[i+m]$.

Deleting the leftmost letter.

We note that the longest repeated prefix of $y[i\,..\,i+m]$ is $y[i\,..\,\tilde{p}_{i+1}]$, and its longest repeated suffix is $y[\tilde{s}_i\,..\,i+m]$. At the beginning of this step they correspond respectively to the head node and the active node. Consider the parent of the oldest leaf of the tree, similarly as in Senft's algorithm two cases are distinguished.

- If the head node is an explicit node, then it is the parent of the oldest leaf. We remove the leaf and its incoming edge. If the head node has only one remaining child, we delete the node and merge the two edges; the *maw*-set associated to the node is added to the leaf.

- Otherwise, the head node is on the edge leading to the oldest leaf. We replace the leaf by a new one labelled by $\tilde{s}_i$, with $y[\tilde{s}_i - 1]$ as the only preceding letter, and the edge is relabelled by $y[\tilde{s}_i - 1]$. We add $y[\tilde{s}_i - 1]$ to the set of preceding letters of the parent of the leaf.

The MAWs associated to the leaf we have deleted were those of type 3 (Lemma 9.4.5). We now update the tree and compute the other MAWs to remove and add.

We visit the oldest leaf in the tree and empty its set of preceding letters. Then we move up in the tree following back the edges until we have covered $\tilde{p}_{i+1} - i$ letters. We move the head node to this node: it corresponds to the factor $y[i+1\,..\,\tilde{p}_{i+1}]$. If the active node was equal to the head node, we move the active node to this node; we have $s_{i+1} = \tilde{s}_i - 1$, otherwise we have $s_{i+1} = \tilde{s}_i$. Each of the explicit nodes visited on the path from the oldest leaf to the head node corresponds to a factor $y[i+1\,..\,j]$, with $p_{i+1} \geq j > \tilde{p}_{i+1}$. For each of them, we remove $y[i]$ from their set of preceding letters. For each of their children, we remove letter $y[i]$ (tagged or not) from their *maw*-set (type 2 Lemma 9.4.5).

There is at most one MAW of type 1 that has to be deleted (Lemma 9.4.5). It exists iff $y[i] = y[i+1]$ and $pp_{i+1} = \tilde{p}_{i+1} + 1$, in which case we remove it from the *maw*-set of the child of the head node by following the edge starting with $y[i]$. According to Lemma 9.4.4, removing the leftmost letter creates one MAW, which is $y[i]y[i+1\,..\,\tilde{p}_{i+1}+1]$, thus we add $y[i]$ to the *maw*-set of the child of the head node by following the edge starting with $y[\tilde{p}_{i+1}+1]$. If the head node is implicit and thus equal to the active node we tag the letter $y[i]$.

Finally, if the head node is above the parent of the oldest leaf of the tree, we move it down to this node. If the active node is implicit and on the edge leading to the oldest leaf of tree we set the head node equal to the active node.

An illustrative example of our algorithm is shown in Section 9.7.

Complexity.

The algorithm extends Senft's algorithm for the construction of the suffix tree in a sliding window. For addition and deletion of a letter, the number of operations is respectively $\mathcal{O}(\sigma(\tilde{s}_i - s_i))$ and $\mathcal{O}(\sigma(p_{i+1} - \tilde{p}_{i+1}))$. Similar to the proof of Theorem 9.4.7, we obtain that the total number of operations is $\mathcal{O}(\sigma n)$. We use $\mathcal{O}(\sigma m)$ space to store the suffix tree for the factor inside the window. The $\mathcal{O}(\sigma)$ factor is to store an array of size σ at each explicit node for constant-time child queries. We also use up to $4m$ arrays of size σ each to store the two sets of letters – the suffix tree has no more than $2m$ explicit nodes. We also store the word itself over two windows. Thus the total space complexity is bounded by $\mathcal{O}(\sigma m)$. We thus obtain the following result.

Theorem 9.5.2. *Given a word of length n on an alphabet of size σ, our algorithm computes the set of minimal absent words in a sliding window of size m in $\mathcal{O}(\sigma n)$ time and $\mathcal{O}(\sigma m)$ space.*

9.6 Applications to on-line pattern matching

Chairungsee and Crochemore introduced the Length Weighted Index (LWI), a metric based on the symmetric difference of minimal absent words sets [84]. The LWI was then applied by Crochemore et al. [85] to devise an $\mathcal{O}(m+n)$-time and $\mathcal{O}(m+n)$-space algorithm for alignment-free comparison of two sequences of length m and n on a constant-sized alphabet. More recently, different such indices have been studied for sequence comparison and phylogeny reconstruction [86]. We base our new pattern matching algorithm on this LWI. To maintain the LWI across the word y, we need to compute the set of minimal absent words in a sliding window of size $m = |x|$ of y. Several linear-time and linear-space algorithms have been proposed to compute the set of minimal absent words [73, 129, 130, 135]. Ota et al. presented an on-line algorithm that requires linear time and linear space [132]. However, to the best of our knowledge, the problem of computing minimal absent words in a sliding window has not been addressed so far.

The LWI is based on the size of the minimal absent words that are in the symmetric difference of the two sets of MAWs we compare. Thus a direct application

to our algorithm is on-line approximate pattern matching using the LWI over the pattern an every window of size m of the text.

This yields, to the best of our knowledge, to the *first* algorithm for the classical on-line exact pattern matching problem that uses some form of negative information (minimal absent words) for the comparison.

Theorem 9.6.1. *Given a word x of length m on an alphabet Σ of size σ, one can find on-line all occurrences of x in a word y of length $n \geq m$ on alphabet Σ in $\mathcal{O}(\sigma n)$ time and $\mathcal{O}(\sigma m)$ space. Within the same complexities, one can also compute on-line $\mathsf{LWI}(M(x), M(y[i\,..\,i+m-1]))$, for all $0 \leq i \leq n-m$.*

Proof. As a pre-processing step, we first build the suffix tree of x and compute the MAWs of x. At the same time, by Lemma 9.5.1, we represent all MAWs of x by storing a set of letters in each explicit node of the tree. This can be done in $\mathcal{O}(\sigma m)$ time and space [73]. We then apply Theorem 9.5.2 to build the suffix tree for a sliding window of size m over y on top of the suffix tree of x. This way when a MAW is created or deleted we can update LWI in $\mathcal{O}(1)$ time as we can check if it is a MAW of x or not. For the first part, note that two words x and z are equal if and only if $\mathsf{LWI}(M(x), M(z)) = 0$ [73, 78]. We thus obtain the result. □

9.7 An illustrative example of the algorithm

Let $y = \overset{0}{\mathsf{A}}\overset{1}{\mathsf{C}}\overset{2}{\mathsf{A}}\overset{3}{\mathsf{C}}\overset{4}{\mathsf{A}}\overset{5}{\mathsf{A}}\overset{6}{\mathsf{G}}\overset{7}{\mathsf{C}}\overset{8}{\mathsf{A}}\overset{9}{\mathsf{G}}\overset{10}{\mathsf{A}}\overset{11}{\mathsf{A}}\cdots$ and $m = 8$. The first three window shifts are illustrated in the figures below. The longest repeated prefixes (head node) are shown in green and the longest repeated suffixes (active node) are shown in blue. The modifications from one tree to the other are shown in red. The sets representing the MAWs are denoted by M and those representing the set of preceding letters are denoted by B. Tagged letters are shown by a subscript '$_+$'. The figures are to be read from left to right and then from top to bottom.

First shift of the window $z_0 = \overset{0}{\mathsf{A}}\overset{1}{\mathsf{C}}\overset{2}{\mathsf{A}}\overset{3}{\mathsf{C}}\overset{4}{\mathsf{A}}\overset{5}{\mathsf{A}}\overset{6}{\mathsf{G}}\overset{7}{\mathsf{C}}$:

- **Part 1**: Adding A on the right. The active node has a child by A, thus there is no MAWs of type 3, the active node is also the node corresponding to the suffix starting at $\tilde{s}_0$.
  - We remove the MAW GCA, stored in the destination of the edge containing the active node.
  - We move down the active node (in blue) of one letter, this node is explicit. We update its set of letters by adding the letter G.

- In each child of the active node we store $y[\tilde{s}_0 - 1] = \mathsf{G}$ in their *maw*-set (type 2).
- $ss_0 \neq \tilde{s}_0 - 1$ thus there is no MAW of type 1.

- **Part 2**: Removing A on the left.
  - The head node corresponds to an explicit node in green. We remove the MAWs stored in the oldest leaf of the tree (in red), CACAC (type 3). The head node has only one remaining child. Thus we make it implicit, and its *maw*-set goes to its leaf. We add its set of letters to its parents.
  - We go to the oldest leaf in the tree, we empty its set of preceding letters. Then we move 6 letters up. We move the head node to this node. There were no explicit nodes on the way, thus there is no MAW of type 2.
  - $pp_1 \neq \tilde{p}_1 + 1$, thus there is no MAW of type 1.
  - The MAW to create is ACAC, we store it in the oldest leaf with the letter A.

Second shift of the window $z_1 = \overset{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8}{\mathsf{CACAAGCA}}$:

- **Part 1**: Adding G on the right.
  - The active node is explicit but it does not have a child by G, thus we create a leaf labelled $s_1 = 7$, with $y[s_1 - 1] = \mathsf{G}$ as a preceding letter. For every letter a preceding the active node, except G we add a as a MAW in the leaf, thus we add A. We move up the active node following the suffix link.
  - The active node is explicit and it has a child by G.
    - We remove the MAW CAG stored in the child of the active node by G.
    - We move down the active node by following G.
    - The active node is now implicit, thus there is only one letter following its corresponding factor, we add $y[\tilde{s}_1 - 1] = \mathsf{C}$ in the *maw*-set of the destination of its edge (type 2), we tag this letter.
    - $ss_1 \neq \tilde{s}_1 - 1$ thus there is no MAW of type 1.

- **Part 2**: Removing C on the left.
  - The head node is explicit, we remove the oldest leaf, and its corresponding MAWs ACAC and GCAC (type 3).

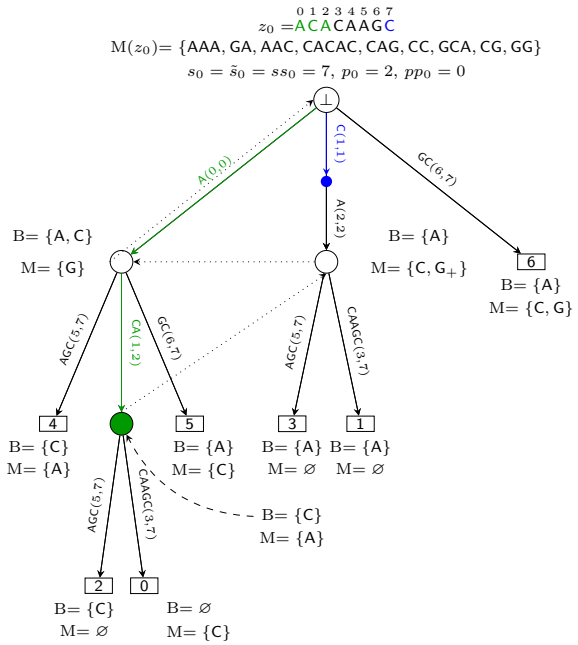
- We visit the oldest leaf of tree. We empty its set of preceding letters. Then we move up 7 letters. We move the head node to this node. There were no explicit nodes on the way, thus there is no MAW of type 2.
- $pp_2 \neq \tilde{p}_2 + 1$ this there is no MAW of type 1.
- There is one MAW to create it is $y[1]y[2..\tilde{p}_2 + 1] = \mathsf{CAC}$ we add it to the head node.

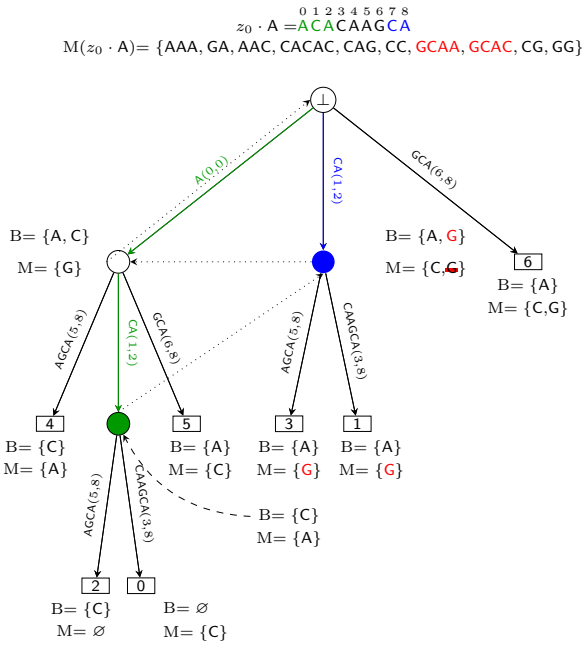
Third shift of the window $z_2 = \overset{2}{\mathsf{A}}\,\overset{3}{\mathsf{C}}\,\overset{4}{\mathsf{A}}\,\overset{5}{\mathsf{A}}\,\overset{6}{\mathsf{G}}\,\overset{7}{\mathsf{C}}\,\overset{8}{\mathsf{A}}\,\overset{9}{\mathsf{G}}$:

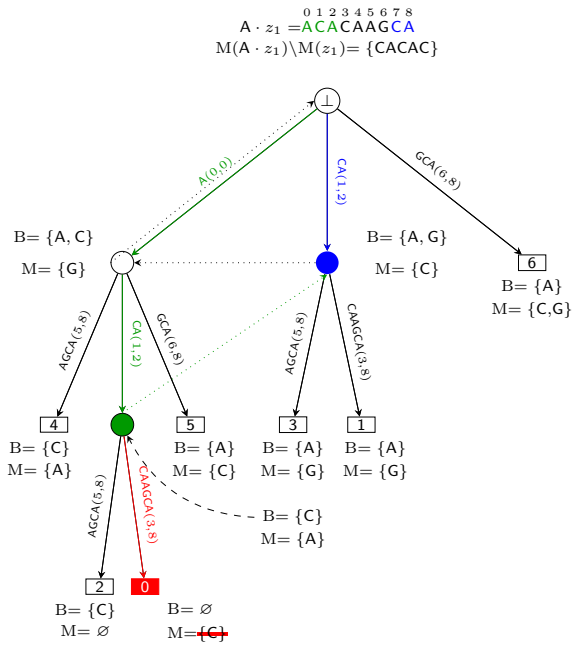
- **Part 1**: Adding A on the right.
  - The active node is implicit, we make it explicit. We set its set of preceding letters equal to its child's. There is no untagged letter in the *maw*-set of its child, we untag the tagged letter of its child.
    - We create a new leaf, labelled $s_2 = 8$ with $y[s_2 - 1] = \mathsf{C}$ in its set of preceding letters.
    - $\mathsf{A} \neq y[s_2 - 1]$ precedes the active node, thus we add A in the *maw*-set of the new leaf (type 3). And we move the active following the suffix links.
  - The active node is still implicit, we make it explicit. We set its set of preceding letters equal to its child's. We move the untagged letters of the *maw*-set of its child to the *maw*-set of the active node.
    - We create a new leaf, labelled $s_2 + 1 = 9$ with $y[s_2] = \mathsf{A}$ in its set of preceding letters.
    - There is no MAWs of type 3, as the active node is only preceded by A. We create a suffix link from the last node created to the active node. We move up the active node by following the suffix links.
  - The active node is now explicit, it is the root:
    - We remove $y[\tilde{s}_2 - 1] = \mathsf{G}$ stored in the child of the active node by A.
    - We move down the active node by following A. This node is explicit, we add $y[\tilde{s}_2 - 1] = \mathsf{G}$ to its set of preceding letters.
    - We add $y[\tilde{s}_2 - 1] = \mathsf{G}$ in the *maw*-set of each child of the active node (type 2).
    - There is no MAWs of type 1.
- **Part 2**: Removing A on the left.

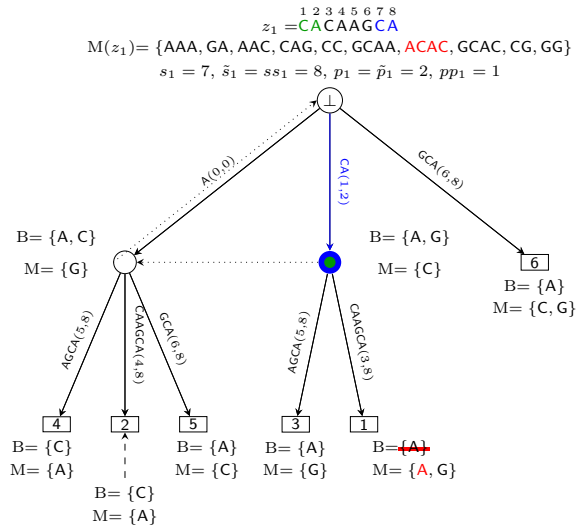
- The head node is explicit, we remove the oldest leaf, and its corresponding MAWs AAC, GAC and GAC (type 3).
- We visit the oldest leaf of tree. we empty its set of preceding letters. Then we move 8 letters up. We move the head node to this node. There is one explicit node on the way, we consider it.
  - We remove $y[2] = \mathsf{A}$ from its set of preceding letters.
  - We remove $y[2] = \mathsf{A}$ from the sets of MAWs of its children.
- There is no MAWs of type 1, there is one MAW to add it is AC, we store it the child of the head node that lead to the oldest leaf in the tree.
- We move the head node to the node corresponding to the parent of the oldest leaf in the tree.

Shift 1: $z_0 = \overset{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7}{\mathsf{ACACAAGC}} \rightarrow z_1 = \overset{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8}{\mathsf{CACAAGCA}}$







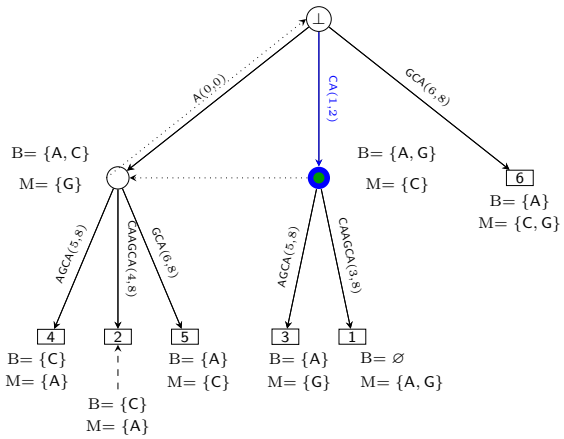


Shift 2: $z_1 = \overset{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8}{\mathsf{CACAAGCA}} \rightarrow z_2 = \overset{2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{\mathsf{ACAAGCAG}}$

$\overset{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8}{z_1 = \mathsf{CACAAGCA}}$

$\mathrm{M}(z_1)$= {AAA, GA, AAC, CAG, CC, GCAA, ACAC, GCAC, CG, GG}

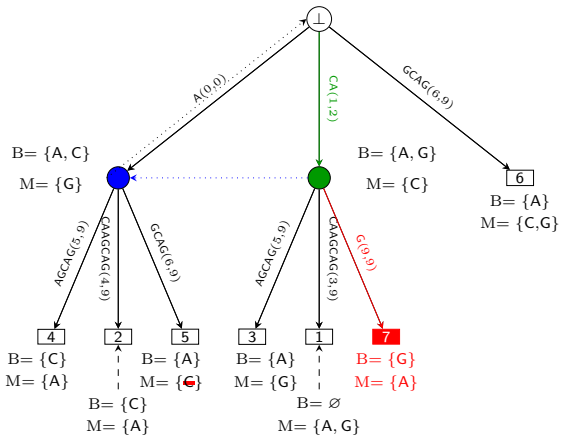
$s_1 = 7,\ \tilde{s}_1 = ss_1 = 8,\ p_1 = \tilde{p}_1 = 2,\ pp_1 = 1$



$\overset{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{z_1 \cdot \mathsf{G} = \mathsf{CACAAGCAG}}$

$\mathrm{M}(z_1) \backslash \mathrm{M}(z_1 \cdot \mathsf{G})$= {CAG}

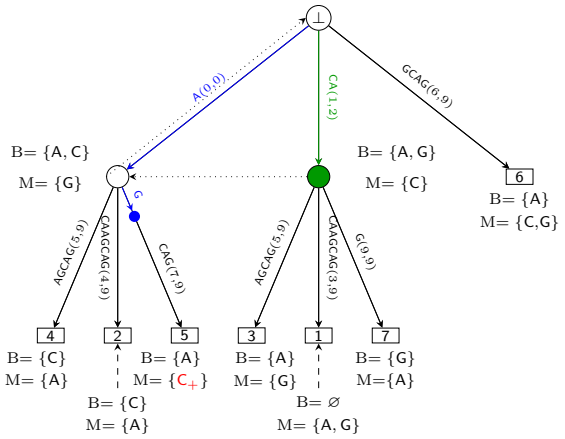
$\mathsf{ACAG} \in \mathrm{M}(z_1 \cdot \mathsf{G}) \setminus \mathrm{M}(z_1)$



$\overset{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{z_1 \cdot \mathsf{G} = \mathsf{CACAAGCAG}}$

$\mathrm{M}(z_1 \cdot \mathsf{G}) \setminus \mathrm{M}(z_1)$ = {ACAG, CAGC}

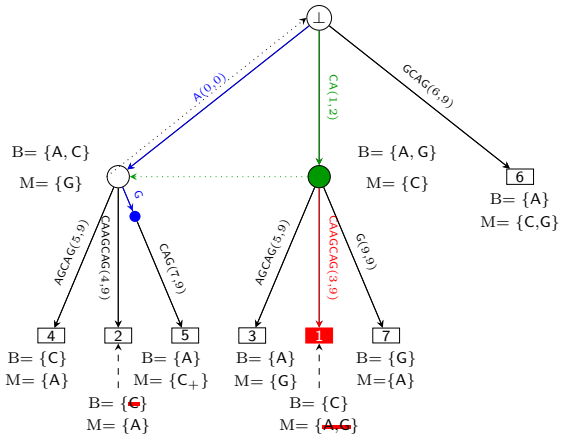
$\mathrm{M}(z_1 \cdot G)$= {AAA, GA, AAC, CAGC, CC, GCAA, ACAC, GCAC, ACAG, CG, GG}



$\overset{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{\mathsf{C} \cdot z_2 = \mathsf{CACAAGCAG}}$

$\mathrm{M}(z_2) \setminus \mathrm{M}(\mathsf{C} \cdot z_2)$ = {CAC}

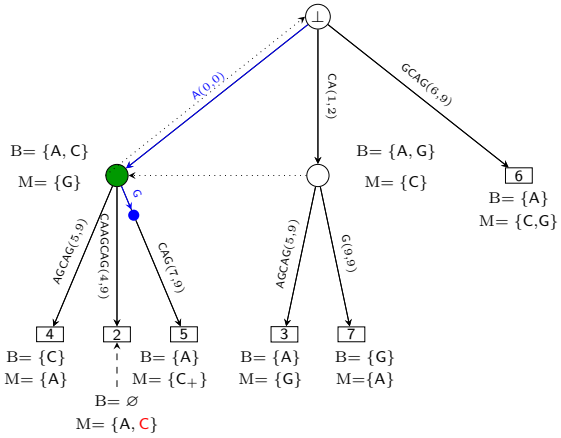
{ACAC, GCAC} $\in \mathrm{M}(\mathsf{C} \cdot z_2) \setminus \mathrm{M}(z_2)$



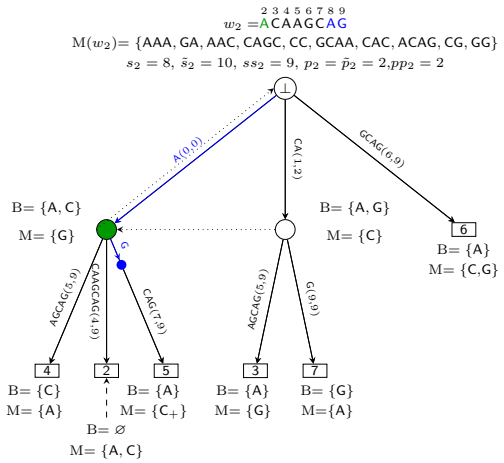
$\overset{2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{z_2 = \mathsf{ACAAGCAG}}$

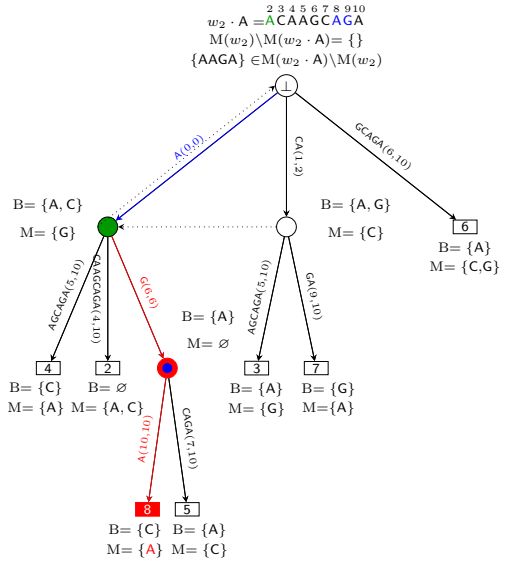
$\mathrm{M}(z_2)$= {AAA, GA, AAC, CAGC, CC, GCAA, CAC, ACAG, CG, GG}

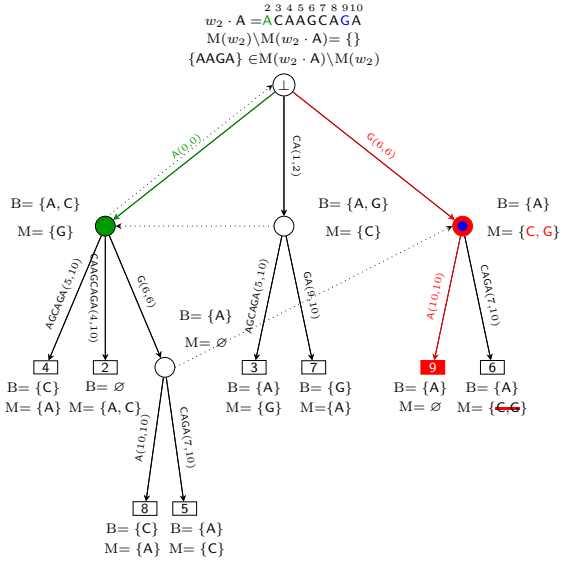
$s_2 = 8,\ \tilde{s}_2 = 10,\ ss_2 = 9,\ p_2 = \tilde{p}_2 = 2, pp_2 = 2$

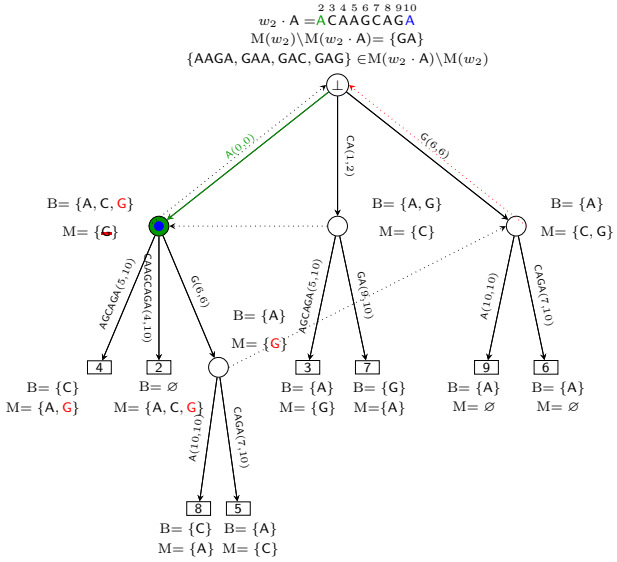


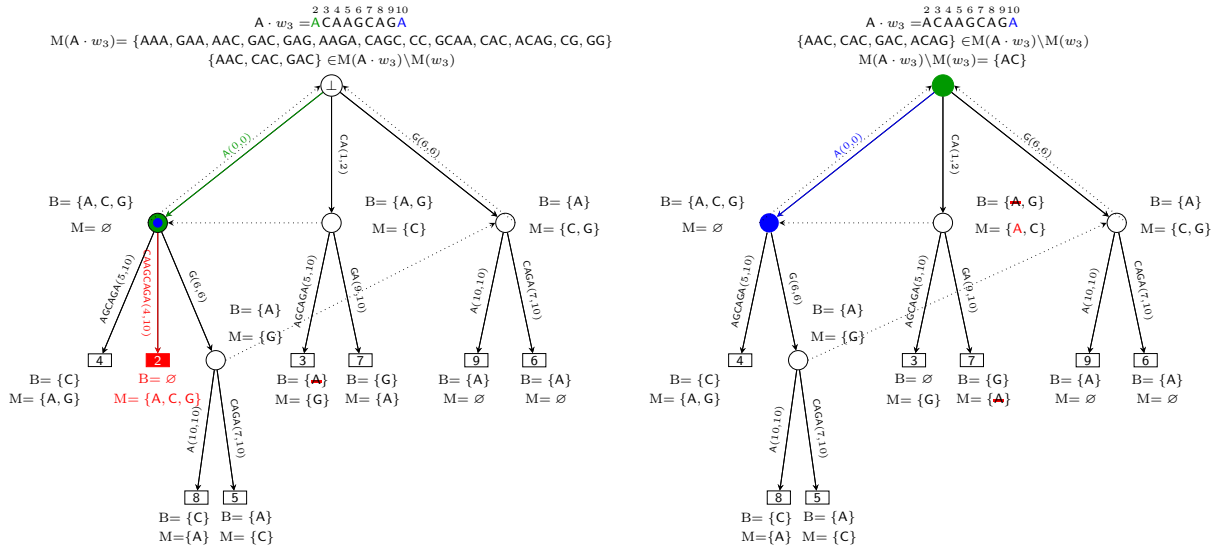
Shift 3: $z_2 = \overset{2\ 3\ 4\ 5\ 6\ 7\ 8\ 9}{\mathsf{ACAAGCAG}} \rightarrow z_3 = \overset{3\ 4\ 5\ 6\ 7\ 8\ 9\ 10}{\mathsf{CAAGCAGA}}$










2 3 4 5 6 7 8 9 10
A · w_3 =ACAAGCAGA
M(A · w_3)= {AAA, GAA, AAC, GAC, GAG, AAGA, CAGC, CC, GCAA, CAC, ACAG, CG, GG}
{AAC, CAC, GAC} ∈M(A · w_3)\M(w_3)
A(0,0)
CA(1,2)
G(6,6)
B= {A, C, G}
M= ∅
B= {A, G}
M= {C}
B= {A}
M= {C, G}
AGCAGA(5,10)
CAAGCAGA(4,10)
G(6,6)
AGCAGA(5,10)
GA(9,10)
A(10,10)
CAGA(7,10)
B= {A}
M= {G}
4
2
3
7
9
6
B= {C}
M= {A, G}
B= ∅
M= {A, C, G}
B= {A}
M= {G}
B= {G}
M= {A}
B= {A}
M= ∅
B= {A}
M= ∅
A(10,10)
CAGA(7,10)
8
5
B= {C}
M={A}
B= {A}
M= {C}
2 3 4 5 6 7 8 9 10
A · w_3 =ACAAGCAGA
{AAC, CAC, GAC, ACAG} ∈M(A · w_3)\M(w_3)
M(A · w_3)\M(w_3)= {AC}
A(0,0)
CA(1,2)
G(6,6)
B= {A, C, G}
M= ∅
B= {A, G}
M= {A, C}
B= {A}
M= {C, G}
AGCAGA(5,10)
G(6,6)
AGCAGA(5,10)
GA(9,10)
A(10,10)
CAGA(7,10)
B= {A}
M= {G}
4
3
7
9
6
B= {C}
M= {A, G}
B= ∅
M= {G}
B= {G}
M= {A}
B= {A}
M= ∅
B= {A}
M= ∅
A(10,10)
CAGA(7,10)
8
5
B= {C}
M={A}
B= {A}
M= {C}
3 4 5 6 7 8 9 10
w_3 =CAAGCAGA
M(A · w_3)= {AAA, GAA, GAG, AAGA, CAGC, AC, CC, GCAA, CG, GG}
$s_3 = \bar{s}_3 = 10,\ ss_3 = 10,\ p_3 = 3\ \bar{p}_3 = 0, pp_3 = 3$
A(0,0)
CA(1,2)
G(6,6)
B= {A, C, G}
M= ∅
B= {G}
M= {A, C}
B= {A}
M= {C, G}
AGCAGA(5,10)
G(6,6)
AGCAGA(5,10)
GA(9,10)
A(10,10)
CAGA(7,10)
B= {A}
M= {G}
4
3
7
9
6
B= {C}
M= {A, G}
B= ∅
M= {G}
B= {G}
M= ∅
B= {A}
M= ∅
B= {A}
M= ∅
A(10,10)
CAGA(7,10)
8
5
B= {C}
M={A}
B= {A}
M= {C}

Conclusion

In the more theoretical part of my PhD, I studied thoroughly the computation of minimal absent words. I proposed four algorithms to compute minimal absent words. The first one, MAW, is the fastest available algorithm to compute all minimal absent words in a sequence over a constant size alphabet. Then pMAW, is another algorithm based on the same idea and engineered to be compatible with parallel computation. em-MAW is adapted to use as little RAM as the user want (up to a few kB). We provided an implementation for these three algorithms, they are available at `https://github.com/solonas13/maw`, under the GNU GPLv3 License.

Finally, I introduced a new algorithm that rather than computing the minimal absent words for the whole sequence, computes them for a sliding window over the sequence. Its applications can be numerous. Indeed when combined with a distance based on the set of minimal absent words it yields to the first algorithm for the on-line exact pattern matching problem that uses some form of negative information for the computation. This problem has many applications including computational biology. Indeed our method could be used to search for seeds in a genome for a mapping algorithm. However this method can not scale because the search is linear in the size of the genome, not in the size of the read as it should be for an aligner. Nonetheless we could try to use it for the reads that did not align on the genome, or to look for a portion of the genome that is similar in terms of absent words. An other application more general to the computation of minimal absent words is the phylogeny. This idea was introduced by Chairungsee in [84], but we could go further by using the algorithms we have developed whose performances are far better from the one used at the time.

I have also worked on the Range Minimal Query (RMQ) problem. It consists in answering efficiently the question: 'what is the minimal element between two specified indices in a given array?'. This problem is highly related to the problem of finding the Lowest Common Ancestor (LCA) of a pair of nodes in a rooted tree, widely studied, especially in phylogeny. We introduced a new approach that is asymptotically slower and less space efficient than state-of-the-art approaches. However our novel structure has a strong advantage; it can be maintained whenever

an insertion, modification or deletion modifies the input sequence. Thus we proposed a new algorithm for dynamic range minimum query [136].

I have had a small contribution in the study of the Combinatorial RNA Design, for the energy models of Watson-Crick and Nussinov-Jacobson [137]. The problem is, given a structure, to find a sequence such that the RNA will preferentially fold into this structure for the chosen energy model. The RNA Design is a core problem because the structure is very important for an RNA, often more than its sequence as the structure gives the function. Solving the RNA Design problem should allow to creates an RNA that will fold into the desired structure. However this problem is very complex, and it requires a realistic energy model. We were able to solve it only for basic energy models and further work need to be done to adapt it to more complex energy models.

These two projects were related but totally independent from the two main projects of my PhD, thus I have not detailed them in the manuscript.

The more applied project of my PhD was to identify the circRNAs in *P.abyssi*. In theory it may seem quite straightforward, but the data revealed to be noisy. Finally, we were able to identified 42 circRNAs with high-confidence because we obtain them by combining the results of several independent experiments. However we are not able to estimate the rate of false negatives and our criteria might be too stringent. That was not our concern as we wanted to have as little false positive as possible to identify the circRNAs that interacts with *Pab1020*. The elaborated method will be used to analyse the output of further RNA-Seq experiments that we have just started and whose aim is to study the circularization function of the Rnl3 protein family in other organisms.

Publications

Peer-reviewed journal articles:

- A. Héliou, S. P. Pissis, S. Puglisi, emMAW: Computing Minimal Absent Words in External Memory, *Bioinformatics*, 2017.
- H.F. Becker, A. Héliou, K. Djaout, R. Lestini, M. Regnier, H. Myllykallio, Hight-Throughput Sequencing Reveals Circular Substrates for an Archaeal RNA ligase, *RNA Biology*, 2017.
- A. Héliou, M. Leonard, L. Mouchard, M. Salson, Efficient dynamic range minimum query, *Theoretical Computer Science*, 2016.
- J. Haleš, A. Héliou, J. Maňuch, Y. Ponty, L. Stacho, Combinatorial RNA Design, *Algorithmica*, 2016.
- C. Barton, A. Héliou, L. Mouchard, S. P. Pissis, Linear-time Computation of Minimal absent Words Using Suffix Array, *BMC Bioinformatics*, 2014.

Conferences with proceedings:

- M. Crochemore, A. Héliou, G. Kucherov, L. Mouchard, S. P. Pissis, Y. Ramusat, Minimal absent words in a sliding window & applications to on-line pattern matching, *FCT 2017* .
- C. Barton, A. Héliou, L. Mouchard, S. P. Pissis, Parallelising the Computation of Minimal Absent Words, *PPAM* 2015.

- A. Héliou, L. Mouchard, W. Smith, Fast Computation of Fibonacci Words, *work in progress.*

Appendix

RNA BIOLOGY
2017, VOL. 0, NO. 0, 1–11
http://dx.doi.org/10.1080/15476286.2017.1302640

Taylor & Francis
Taylor & Francis Group

RESEARCH PAPER

High-throughput sequencing reveals circular substrates for an archaeal RNA ligase

Hubert F. Becker[a,b], Alice Héliou[a,c], Kamel Djaout[a], Roxane Lestini[a], Mireille Regnier[c], and Hannu Myllykallio[a]

[a]LOB, Ecole Polytechnique, CNRS, INSERM, Université Paris-Saclay, Palaiseau, France; [b]Sorbonne Universités, UPMC Univ Paris 06, 4 Place Jussieu, Paris, France; [c]LIX, Ecole Polytechnique, CNRS, Université Paris-Saclay, INRIA, Palaiseau, France

ABSTRACT

It is only recently that the abundant presence of circular RNAs (circRNAs) in all kingdoms of Life, including the hyperthermophilic archaeon *Pyrococcus abyssi*, has emerged. This led us to investigate the physiologic significance of a previously observed weak intramolecular ligation activity of *Pab*1020 RNA ligase. Here we demonstrate that this enzyme, despite sharing significant sequence similarity with DNA ligases, is indeed an RNA-specific polynucleotide ligase efficiently acting on physiologically significant substrates. Using a combination of RNA immunoprecipitation assays and RNA-seq, our genome-wide studies revealed 133 individual circRNA loci in *P. abyssi*. The large majority of these loci interacted with *Pab*1020 in cells and circularization of selected C/D Box and 5S rRNA transcripts was confirmed biochemically. Altogether these studies revealed that *Pab*1020 is required for RNA circularization. Our results further suggest the functional speciation of an ancestral NTase domain and/or DNA ligase toward RNA ligase activity and prompt for further characterization of the widespread functions of circular RNAs in prokaryotes. Detailed insight into the cellular substrates of *Pab*1020 may facilitate the development of new biotechnological applications *e.g.* in ligation of preadenylated adaptors to RNA molecules.

ARTICLE HISTORY
Received 11 October 2016
Revised 13 December 2016
Accepted 28 February 2017

KEYWORDS
Archaea; circular RNA; computational biology; RNA ligase; RNA-Seq

Introduction

For a long time, circular RNA molecules (circRNAs) lacking 3′ or 5′ termini were considered an unusual form of nucleic acids found in few viroids or viruses using a single-stranded RNA molecule as genetic material, participating in maturation of some tRNA genes, or, alternatively, a result of aberrant RNA splicing (for recent reviews, see[1-3]). However, numerous recent genome-wide experimental and computational studies (RNA-seq analyses) tailored toward detection of circRNAs have revealed this class of RNA molecules as abundant in eukarya, including humans. Evolutionary conservation of circRNAs has suggested that they are functionally important. Indeed, recent studies have revealed that circRNAs are differentially expressed in different human cell lines and tissues, serve as regulators of transcription and protein expression and can act as miRNA sponges.[4-7] Several different types of RNA circles originating from diverse cellular processes have been identified. Most eukaryotic circRNAs result from splicing reactions that are catalyzed either by the spliceosome or ribozymes corresponding to Group I and Group II introns. The process called "backsplicing" that connects a downstream splice donor site (5′ splice site) to an upstream acceptor site (3′ splice site) is the most common mechanism producing circRNAs in eukaryotic cells. During the formation of circRNAs either 2′–5′ or 3′–5′ linkages have been detected.[8,9] The resulting molecular "rings" or RNA circles resist degradation by exoribonucleases that require free termini and/or may have increased melting point in comparison to linear nucleic acid molecules. Circularization of RNA molecules may thus drastically influence the structure and/or shape of these molecules, presumably reflecting structural constraints brought about by circularization.

Although the majority of recent work on circRNAs has been performed using human cell lines, circular RNAs also exist in archaea and bacteria, in addition to the aforementioned viroids and RNA viruses. This raises the question how circular RNA molecules are formed in prokaryotes, where RNA splicing is a rare phenomenon. RNA-seq methodology revealed an abundant genome-wide presence of circRNAs in the transcriptome of the archaeon *Sulfolobus solfataricus*[10] such as tRNA introns and rRNA processing intermediates, as well as several protein-coding genes, and many smaller non-coding RNAs (*e.g.*, box C/D RNAs). This genome-wide study is also in agreement with earlier studies that have revealed the presence of circular forms in excised tRNA-introns,[11,12] introns of rRNAs,[13,14] rRNAs processing intermediates[15] and box C/D RNAs in archaea.[16] Note that box C/D RNAs guide site-specific modification (2′-0-methylation) of rRNA and small nuclear RNA in eukaryotes and tRNA in archaea.[17] RNA-seq data obtained using the "minimal" archaeon *Nanoarchaeum equitans* also revealed circular box C/D RNAs sequences.[18] The specific case where the cleaved-out intron contains a C/D box RNA that guide 2′-O-methylation on nucleotides in the anticodon loop of mature tRNA-Trp should also be highlighted.[12,19] At least 2 separate enzymes, an endonuclease and a ligase, are known to be involved in pre-tRNA splicing in archaeal cell free

CONTACT Hubert F. Becker hubert.becker@polytechnique.edu LOB, E. Polytechnique, Route de Saclay, Palaiseau, France, 91128; Hannu Myllykallio hannu.myllykallio@polytechnique.edu Laboratory of Optics and Biosciences, CNRS-INSERM-E, Polytechnique, Route de Saclay, Palaiseau 91128f, France.
Supplemental data for this article can be accessed on the publisher's website.

extracts.[20] The endonuclease cleaves in a characteristic bulge-helix-bulge structure (BHB) of pre-tRNA-Trp, producing 2′-3-cyclic phosphate and 5′-hydroxyl termini joined by the ligase.[11] In addition to ligation of resulting tRNA halves, circularization of the pre-tRNA Trp intron occurs in a mechanistically similar ligation reaction, as observed using *H. volcanii* extracts.[11,19]

Considering the widespread presence of circRNAs in the third domain of life is now evident, further studies on archaeal RNA ligases are now necessary. Up to date, 2 evolutionary unrelated families of RNA ligases capable of joining single stranded RNA molecules have been identified in archaea. RtcB proteins seem to function in most, if not all, archaea as GTP-dependent tRNA-splicing ligases and join spliced tRNA molecules halves to form mature-sized tRNAs molecules *e.g.*, in archaeal precursor tRNA-Trp.[21] The observation that the genomes of some archaea contain 2 open reading frames, previously predicted to function as DNA ligases, led to the discovery of a putative second family of archaeal RNA ligases.[22,23] The founding member of this putative RNA ligase family is the *Pyrococcus abyssi* reading frame *Pab*1020. This family (InterPro code IPR001072) currently contains 170 archaeal and 35 bacterial homologs but little is known concerning the molecular function of this conserved protein family. We have previously shown that, unexpectedly, *Pab*1020 catalyzed the circularization of physiologically non-significant oligoribonucleotides in an ATP-dependent reaction.[22] Similar results have also been reported for closely related *Methanobacterium* protein that catalyzes the intramolecular ligation of 5′-P single-stranded RNA to form a covalently closed circular RNA molecule.[24-27]

Here using a combination of biochemical, RNA-seq and computational analyses, we have investigated the molecular function of the *Pab*1020 RNA ligase family.[24,28] Our genome-wide RNA co-immunoprecipitation studies, using affinity purified *Pab*1020 antibodies, led to the discovery of a large number of circular RNAs that specifically interact with *Pab*1020 RNA ligase in *P. abyssi* cells and were indeed efficiently circularized by the *Pab*1020 *in vitro*. Our studies also indicate a widespread importance of circular RNAs in prokaryotes and suggest a functional speciation of an archaeal polynucleotide ligase toward RNA circularization activity in many thermophilic archaea and bacteria. The identification of physiologic substrates for the *Pab*1020 RNA ligase may also facilitate development of new biotechnological applications for this enzyme family currently commercialized for *e.g.*, labeling of 3′ termini of RNA.

Results

Domain structure of the Pab1020 family RNA ligases

*Pab*1020 is the founding member of the conserved Rnl3 family of RNA ligases[28] that are predominantly found in hyperthermophiles (archaea, bacteria) and halophiles. Each *Pab*1020 monomer consists of 4 domains: the amino-terminal (N-term), the catalytic nucleotide transferase (NTase), the dimerization (Dim) and the C-terminal (C-term) domains [[22], Fig. S1A]. To address why these proteins are frequently annotated as "DNA" ligases, we have performed several sequence similarity searches indicating that the central NTase domain of *Pab*1020 is closely related to the CDC9 domain found in ATP-dependent DNA ligases [residues 66–235, E-value 1.21^{-5}, CDD database[29]]. This domain carries the conserved nucleotide binding domain and corresponds to the minimal catalytic core of this family of enzymes. More sensitive HHpred searches based on the pairwise comparison of profile hidden Markov models (HMMs) also indicated that this domain is related to COG1793 [ATP-dependent DNA ligases, E-value = 7.6^{-23}) and COG0272 [NAD-dependent DNA ligases, E-value = 2.1^{-14}]. Note also that the NTase domain of *Pab*1020 is 26% identical and 37.8% similar with the corresponding domain from the experimentally validated *P. abyssi* DNA ligase *Pab*2002 and carries all the expected motifs for ATP-dependent polynucleotide ligases.[22] Additional HHpred searches using the *Pab*1020 protein sequence as a query further indicated that, at the sequence level, this proteins is related to ATP or NAD+-dependent DNA ligases, as well as different families of RNA ligases, mRNA capping enzymes and RNA repair enzymes (Table 1). These results explain why members of the Rnl3 family are frequently misannotated as DNA ligases. An excellent example of this is the putative *Aquifex aeolicus* DNA ligase (PDB code 3qwu_A in Table 1). This bacterial protein is very likely an RNA ligase, as it contains a dimer interface that is only conserved among the Rnl3 family members in the PDB databank.

Differently from monomeric DNA ligases, Pab1020 (Rnal3) ligases form a homodimer, which is a very rare feature among polynucleotide ligases and warrants further attention. Several features predict that this dimer interface is critical for *Pab*1020 complex assembly and/or function. For instance, PDBePISA analysis showed that the interface formed between 2 monomers has an interface area of ~2274 Å^2, with a high complex formation significance score of 0.518. Up to 58 residues (15.5% of total) interact between the 2 *Pab*1020 monomers. Interestingly, these residues are not randomly located within the *Pab*1020 polypeptide but, on the level of the primary sequence, are mainly found on either side of the catalytic domain.[22,24] One particularly interesting interface residue is Gly296 that is strictly conserved among the *Pab*1020 family members (several additional residues of the domain interface are also evolutionary conserved). This residue is part of a C-terminal dimerization domain composed of 3 α-helices found at the dimer interface and generates a kink in the helix α10. Our dynamic light scattering studies indicated a hydrodynamic radius of approximately 10 nm for the *Pab*1020 wild type protein while the mutants G296A and ΔC-ter formed higher molecular weight molecules without precipitating. Therefore we suppose that the dimerization and C-terminal domains of *Pab*1020 are required for optimal assembly of the active homodimer. A similar suggestion has very recently been made for the *M. thermoautotrophicum* RNA ligase.[25] Note also that the C-terminal domain of *Pab*1020 forms additional contacts with the other monomer, particularly in the proximity of the active site.

Table 1. Level of sequence similarity of *Pab*1020 with different protein families carrying the nucleotidyltransferase (NTase) domain.

Hit	PDB code (chain)	Description of the hit[a]	E-value	number of aligned positions	Query range	HMM Template
		HMM-HMM comparison against protein structure database				
1	2vug_A	*Pab*1020; RNA ligase	5.6E-63	381	1–382	8–389
2	5d1p_A	ATP-dependent RNA ligase	2.2E-51	366	9–380	7–378
3	3qwu_A	Putative DNA ligase	1.0E-50	361	17–382	4–370
4	1dgs_A	DNA ligase; NAD-dependent	1.3E-38	270	37–344	32–369
5	4 glx_A	DNA ligase, NAD-dependent	6.0E-38	272	37–344	30–371
6	2owo_A	DNA ligase, NAD-dependent	1.7E-37	272	37–344	30–371
7	3sgi_A	DNA ligase, NAD-dependent	2.3E-38	272	37–344	39–381
8	1b04_A	DNA ligase, NAD-dependent	2.1E-31	224	37–284	32–317
10	3jsl_A	DNA ligase; NAD+-dependent	6.7E-30	219	37–260	30–310
14	5cot_A	*Naegleria gruberi* RNA ligase	1.1E-16	173	62–244	133–330
15	1s68_A	RNA ligase 2; ATP-dependent	5.7E-16	174	65–247	2–231
16	2hvq_A	T4 RNA Ligase 2, ATP-dependent	1.0E-15	199	65–276	3–272
17	2c5u_A	RNA ligase, ATP-dependent	2.8E-14	200	29–244	30–244
18	3l2p_A	DNA ligase 3, ATP-dependent	1.6E-13	159	87–246	246–426
19	3w1b_A	DNA ligase 4, ATP-dependent	2.3E-13	181	65–246	241–455
20	2cfm_A	*Pyrococcus* DNA ligase (ATP)	2.9E-13	157	87–245	241–424
23	4pz7_A	mRNA-capping enzyme	6.9E-14	202	43–259	21–259
27	1ckm_A	mRNA capping enzyme	2.3E-12	165	66–252	54–245
28	1xdn_A	RNA editing ligase MP52	3.7E-12	173	65–244	8–263
29	3rtx_A	mRNA-capping enzyme	7.5E-13	202	45–264	25–257
31	1p16_A	GTP–RNA, mRNA capping	2.3E-13	205	43–259	19–259
32	1a0i_A	DNA ligase; ATP-dependent	6.5E-12	152	88–246	26–242
33	3kyh_C	mRNA-capping enzyme	2.3E-12	204	44–259	23–267
34	4xrp_B	RNL; RNA repair, kinase	1.2E-06	199	43–244	5–316

[a]The level of sequence similarity of *Pab*1020 to the different proteins carrying a NTase domain was revealed using HHpred implemented at www.toolkit.tuebingen.mpg.de. Homologies were detected using a highly sensitive HMM-HMM comparison mode with default settings. Searched database used was "pdb70_12jun16. Similar results were obtained using Pfam and COG databases.

Polynucleotide ligase, but not nucleic acid binding, activity of Pab1020 is specific for RNA

The nucleic acid binding activity of *Pab*1020 was investigated by electrophoretic mobility shift (EMSA) assays using various Cy5-labeled nucleic acid substrates. Binding reactions were performed using 5′-dephosphorylated substrates to prevent any substrate circularization and/or ligation activities during EMSA assays (Fig. 1A). Our results revealed that *Pab*1020 formed a well-defined oligonucleotide-ligase complex with ssRNA oligonucleotides under stoichiometric conditions. Under these experimental conditions, some ssDNA binding was also observed (Fig. 1A). We then investigated the ligase activity of *Pab*1020 at 50°C after 5′-phosphorylating oligos used for binding assays, including Mn^{2+} as divalent cation in a standard activity buffer. Our results revealed that *Pab*1020 only showed circularization activity on ssRNA, while no activity was observed with ssDNA (or the different RNA/DNA homo- or hetero- oligonucleotide duplexes tested) used for binding assays (Fig. 1B). In these experiments, the circular RNA product migrated "faster" than the linear substrate ssRNA, in agreement with earlier studies.[25] In these assays, the "catalytic" K95G mutant of RNA ligase *Pab*1020 was inactive, thus confirming that RNA specific circularization was indeed catalyzed by *Pab*1020. The *Pab*1020 G296A variant and the NTase domain alone still had weak RNA binding activity (Fig. 1C). We also stress that *Pab*1020 mutants G296A and "NTase domain" were inactive in circularization assays, albeit they still possessed a very weak adenylating activity, as witnessed by a very faint band migrating "slower" than the linear RNA substrate (Fig. 1D). These results suggest importance of the dimerization and/or the C-terminal domains for RNA circularization activity.

Pab1020 interacts with circular RNAs in cell-free extracts

As our biochemical studies confirmed that *Pab*1020 indeed acts as RNA ligase, we further investigated the substrate specificity of this protein in the cell. For these studies, 2 different RNA samples were analyzed using an experimental and computational RNA sequencing pipeline (Fig. 2A). For these studies a total RNA sample was extracted from a stationary phase culture of *P. abyssi* while a second sample was obtained by co-immunoprecipitation of RNA ligase *Pab*1020 after formaldehyde crosslinking between *Pab*1020 and cellular RNAs (RIP assay). Note that the affinity purified polyclonal antibody used for pull-down experiments revealed a single band in Western immunoblots of *P. abyssi* cell-free extracts (Fig. S1B). Isolated RNase III-fragmented RNA was used to prepare a RNA-seq library and sequenced following the Ion Torrent PGM RNA-seq protocol. Typical sequencing runs yielded ~400 000 reads with a read size of 80 to 90 base pairs. All these reads were mapped to the *P. abyssi* reference genome reference (NC_000868, 1765118 base pairs) using Blastn. The unique criterion of the inversion of 2 matches within the same locus (Fig. 2A) led to the identification of approximately 80 000 putative circular RNAs suggesting up to 30 000 distinct junctions. To enrich for circRNAs, we used RNase R that specifically degrades linear RNA molecules in a 5′-3′ direction. For the total RNA fraction, our data revealed approximately 285 000 reads that resisted RNase R treatment and were mapped to the genome. Note that we still obtained linear reads using RNase R treated samples indicating that RNase R did not degrade all linear RNA molecules under these conditions (Table 2). However, 11 to 15 % of reads obtained using an RNase R enrichment were classified as "circular" using our computational criteria (see material and methods). These circular reads covered only a minor part of the transcribed genome

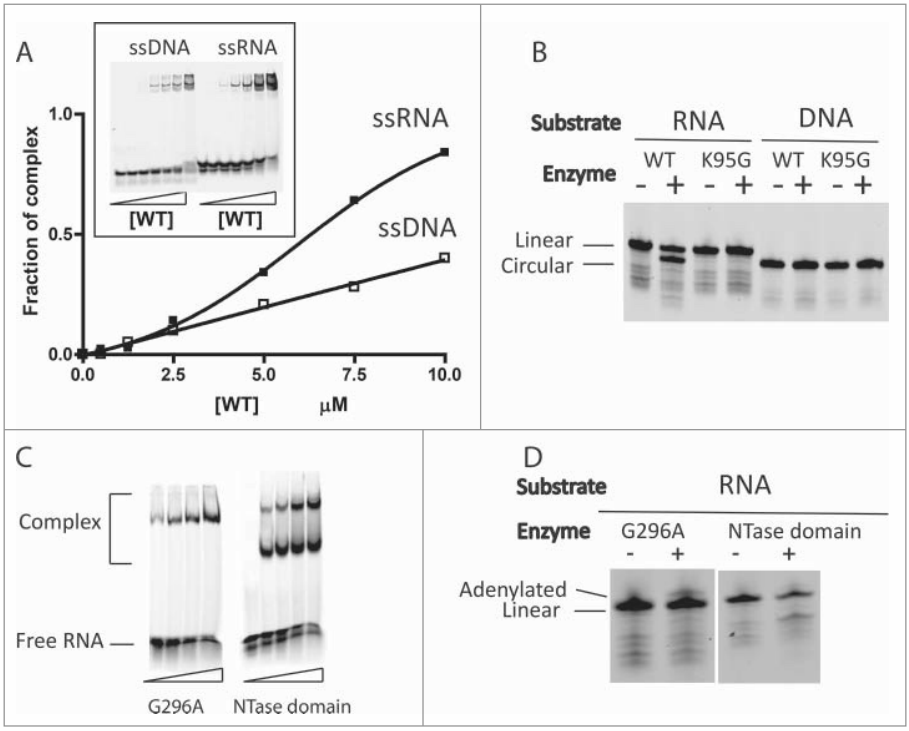


Figure 1. *Pab*1020 RNA ligase binds single-stranded DNA and RNA, but only circularizes single-stranded RNA oligonucleotides. (A) EMSA assays were performed with internally labeled (Cy5) single stranded DNA or RNA oligonucleotides using increasing amounts of wild-type (WT) *Pab*1020 RNA ligase. The relative amount of bound DNA or RNA was plotted against the protein concentration. Insert: On the EMSA gel, the amount of the higher molecular weight bands, corresponding to *Pab*1020-nucleic acid complexes, increased as a function of the protein concentration. (B) RNA and DNA ligation assays with WT and mutant K95G of *Pab*1020 RNA ligase. Standard ligation reactions containing 10 pmol Cy5-RNA or -DNA molecules and 200 pmol RNA ligase *Pab*1020 were incubated 90 min at 50°C. Reaction products were resolved on denaturing PAGE and a 700 nm scan of the gel was performed on Licor Odyssey Infrared Imager. While no activity was observed with DNA substrate, *Pab*1020 RNA ligase circularized an RNA oligoribonucleotide as shown on the gel with the apparition of a lower band corresponding to circular RNA molecules. Expectedly, a control reaction with an inactive enzyme (mutant K95G) presented no lower band. (C) Identical to panel (A), except that the enzymes used in the EMSA assays corresponded to the mutant G296A (dimerization domain) and the amino-terminal domain of 250 residues carrying a nucleotide transferase (NTase) domain. Both mutants were able to form RNA-Protein complexes with 18-mers single-stranded RNA. (D) Identical to panel (B), except that circularization was performed only with RNA substrate and with G296A mutant and NTase domain. No circRNAs were observed (positive control is indicated in panel B).

(Table 2), therefore indicating that the combined experimental and computational criteria are strict.

We are aware that our experimental and data analyses protocols may be prone for unwanted artifacts. Hence, to establish more selective criteria for circRNA identification we first merged the sequencing data from all of our experiments to identify the maximum number of circular RNAs. We observed that the circularization junctions were frequently shifted by 1 to 3 nucleotides between the different reads. This may either reflect the slight heterogeneity in choosing the transcription initiation site, or, alternatively, the presence of an identical base in 5′ and 3′ termini of a transcript that cannot be solved during the read mapping. Thus, we grouped together those putative circular reads where the circular junctions were located within 3 nucleotides. To be classified as circular, we only selected junctions that were identified at least in 2 independent experiments and supported each time by more than 3 individual reads that may have different start and end positions. We also requested that at least half of these putative circular reads that aligned entirely inside a given putative junction supported the junction.

With these strict and multiple constraints, we identified in *P. abyssi* a total of 133 individual circRNA loci (Fig. 2B) supported by 28 279 circular reads (Fig. 2C). The large majority of these RNA molecules interacted with Pab1020 in pull down experiments (Table 2, see also discussion).

Identification of functional categories of circular RNAs

The 133 *P. abyssi* circRNA loci represent 5 distinct functional groups: C/D Box RNA, non-annotated small RNAs, protein coding RNA, tRNA and rRNA (Fig. 2B). Among these, circular reads were overrepresented for 38 circular C/D Box and 5 tRNA molecules. Although 71 loci out of 133 loci corresponded to protein coding mRNAs, these were supported only by 3% of the analyzed reads (Fig. 2C). This approach also led to the discovery of 13 new circular RNAs was also identified [marked as non-annotated (NA) in Fig. 2B and C].

As pointed out by Danan *et al*,[10] it is necessary to be attentive in attributing positions of circular junctions that are possibly influenced by reverse transcription, sequencing and mapping errors. The constraints applied in our computational pipeline (see materials and methods), allow attaining highly selective circRNAs identification. Numerous circRNAs, including non-coding RNAs, i.e. C/D box RNA, tRNA-intron and rRNA, were also observed in previous study.[10]

We also noticed that circRNAs corresponding to the different functional categories did not behave identically in RNase R-enrichment experiments. Strikingly, the relative portion of circular reads markedly increased after RNase R treatment from 35% to 86% for C/D Box RNAs and was constantly high, around 88%, for tRNAs (Fig. 2D). The fact that RNase R

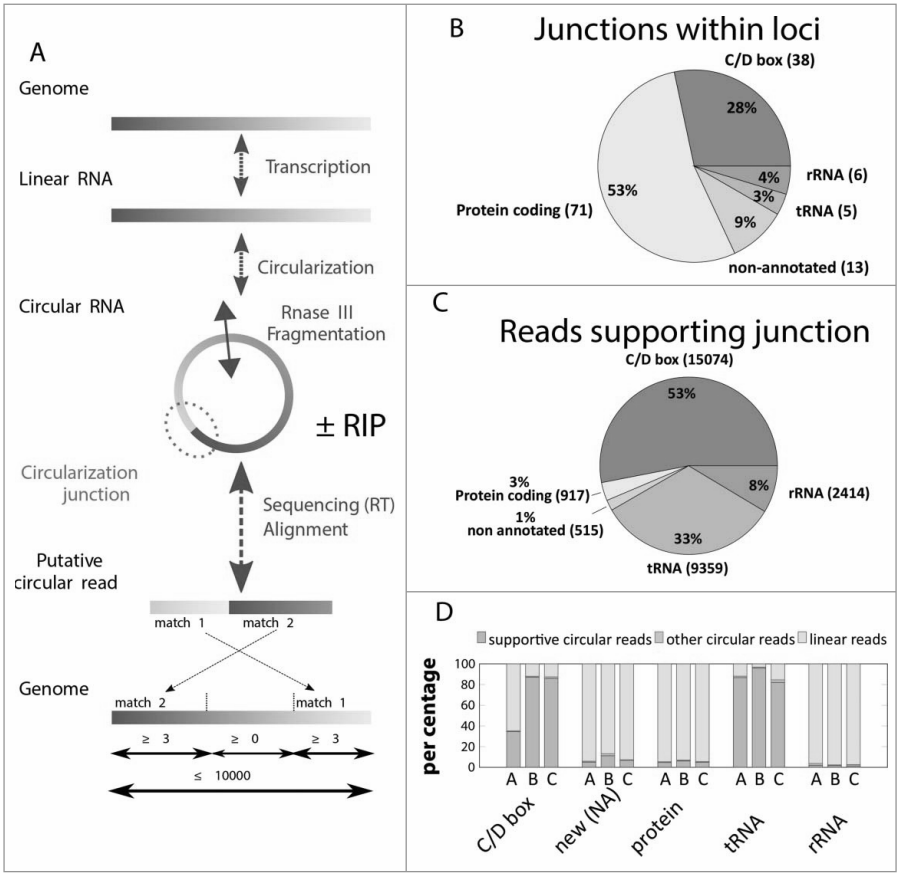


Figure 2. Identification of *P. abyssi* circRNAs using high throughput sequencing. (A) The workflow for identification of circularization junctions using RNA samples isolated from *P. abyssi* cells using IonTorrent semiconductor-based sequencing technology is shown. " ± RIP" refers to the fact that identical computational approach was used for total and RNA immunoprecipitation (RIP) samples. Obtained linear and circular RNA molecules were fragmented at least once (indicated by a double arrow in panel A) using RNase III treatment. Following reverse transcription, samples were sequenced and obtained reads were aligned to the *P. abyssi* reference genome using Blastn. Reads were considered circular if 2 permuted matches covering the whole read was detected. (B) Number and percentage of the different functional classes (*loci*) considered circular in our sequencing experiments. (C) Number and percentage of the sequencing reads (total 28 279) supporting circularization of the different functional groups. (D) Percentage of the reads supporting RNA circularization (supportive circular reads) of the different RNA categories. Only intron containing tRNAs as identified as circular were included in the analysis. "Other circular reads" refers to a minority of putative circular reads that fulfill all the computational criteria indicated in panel A without supporting the junctions identified in panel B. Samples used were: A, circular reads after RIP assays using *Pab*1020 antibodies; B, circular reads after RIP assay and ribonuclease R treatment; C, circular reads in total RNA samples treated with ribonuclease R. New (NA) refers to previously non-annotated loci.

treatment induces an enrichment in the amount of reads supporting circularization junctions in pull-down and total RNA samples further indicates that *Pab*1020 RNA ligase specifically associates with circular RNA loci in *P. abyssi* cells. For the 3 additional functional groups, this RNase R enrichment for circRNAs was less obvious (Fig. 2D). Note that for the specific case of the tRNA-Trp, the circularization of the encoded-intron occurs simultaneously during the splicing process and linear intron intermediates is not expected to occur. For others RNAs (NA, protein coding and rRNA), the amount of circular reads is too low compared with linear reads for a same locus to allow the enrichment visualization. However, 3 non-annotated circRNAs (NA7, NA12, NA13 in Table 3) out of 13 showed some enrichment supported by significant amount of reads (Table 3). A high number (∼38 000) of circular reads were mapped to rRNAs (5S, 7S, 16S and 23S rRNAs) but in most cases, localization of the precise position of the junction point from permutated reads was far from evident, possibly reflecting the length and highly structured nature of these RNAs that hinders activity of the reverse transcriptase. However, in the case of the 5S

Table 2. The summary of the RNA-seq results and RIP assays using the *Pab*1020 antibody.

		Circular reads		Linear reads		
Abbreviation	Sample	% of mapped reads	% of genome[a]	% of mapped reads	% of genome	Number of mapped reads
A	RIP	8.3	6.8	91.7	50.3	247060
B	RIP + RNase R	14.7	2.7	85.3	8.7	44413
C	Total RNA + RNase R	11.4	2.8	88.6	7.3	284856

[a]The genome size of *P. abyssi* is 1. 76 Mbp of which at least 79.5% is transcribed. The portion of the genome (% of genome) covered by the mapped reads is indicated in each case. Samples are referred to using abbreviations "A," "B" and "C" in the Figs. 2 and 4.

Table 3. List of 42 highly significant circular RNA molecules interacting with *Pab*1020 RNA ligase in cells. A summary of these results in the form of a Venn diagram is presented in Fig. 4.

Locus	CircularRNA Start	End	Size	Enrichment[a]
38 C/D box RNAs				
sR46	57374	57441	67	1.05
*Pab*snRNA44 (sR45)	64244	64306	62	7.08
*Pab*snRNA21 (sR14)	65218	65278	60	2.87
sR22	65333	65397	64	2.41
sR32	67951	68012	61	1.97
*Pab*snRNA3 (sR21)	127067	127128	61	15.73
*Pab*snRNA10 (sR2)	230632	230696	64	1.94
sR49	235439	235502	63	1.29
*Pab*snRNA33 (sR13)	245974	246035	61	3.86
sR31	258066	258127	61	2.16
*Pab*snRNA35 (sR29)	318118	318182	64	2.81
*Pab*snRNA32 (sR4)	473175	473236	61	52.2
*Pab*snRNA38 (sR58)	541770	541831	61	2.15
*Pab*snRNA40 (sR39)	543855	543917	62	4.6
*Pab*snRNA31 (sR20)	553656	553721	65	2.67
*Pab*snRNA12 (sR26)	631518	631581	63	5.74
*Pab*snRNA39 (sR60)	631584	631644	60	7.49
*Pab*snRNA13 (sR44)	636702	636762	60	13.58
*Pab*snRNA17 (sR7)	648165	648228	63	43.0
sR25	675408	675468	60	62.5
*Pab*snRNA36 (sR55)	910497	910569	72	4.91
*Pab*snRNA27 (sR35)	949199	949261	62	2.83
sR56	960309	960370	61	5.31
*Pab*snRNA25 (sR3)	991446	991505	59	13.74
*Pab*snRNA1 (sR24)	1026074	1026133	59	6.93
sR53	1042251	1042319	68	1.11
*Pab*snRNA46 (sR38)	1065728	1065791	63	5.14
*Pab*snRNA28 (sR37)	1195780	1195842	62	2.32
*Pab*snRNA23 (sR1)	1209270	1209329	59	36.27
*Pab*snRNA34 (sR59)	1260126	1260195	69	3.21
sR41	1292356	1292415	59	5.63
*Pab*snRNA5 (sR11)	1397126	1397188	62	28.62
*Pab*snRNA29 (sR8)	1403662	1403723	61	113.04
*Pab*snRNA42 (sR36)	1409130	1409196	66	9.73
*Pab*snRNA41 (sR48)	1468649	1468712	63	1.23
*Pab*snRNA6 (sR6)	1536388	1536449	61	4.33
*Pab*snRNA45 (sR34)	1755871	1755930	59	3.02
*Pab*snRNA9 (sR12)	1755929	1755991	62	6.88
1 tRNA (intron)				
*Pab*t35	1330513	1330584	71	0.89
3 non-annotated (NA) new circular RNAs				
NA7	622461	622526	65	8.39
NA12	1011096	1011158	62	1.91
NA13	1674520	1674581	61	5.36

[a]An enrichment factor for circular molecules was estimated using the ratios of the circular and linear reads in the pulldown fraction (PA) and the total RNA (PC) samples before and after RNase R treatment. The values shown were obtained by dividing PA by PC. Values higher than 1 indicate the enrichment for circRNAs due to selective degradation of linear RNA molecules.

rRNA, we identified 170 permuted reads indicating a specific circularization event between the 5′ and 3′ extremities of 5S rRNA (with a 10 nucleotide margin). As 5S rRNA interacted with *Pab*1020 in cell-free extracts and its circular form has been previously observed,[10] this enzyme may participate in 5S rRNA pre-processing *via* a circular intermediate, as previously proposed for 16S and 23S rRNAs.[15]

Circularization of physiologically significant RNA molecules by an archaeal RNA ligase Pab 1020

To test whether RNAs interacting with *Pab*1020 in the cells may correspond to physiologically significant substrates of this enzyme, we assayed the ligation activity of *Pab*1020 RNA ligase using the linear fluorescent Cy5-RNA transcripts for 3 circRNAs identified during this work (Fig. 3). For these biochemical studies, we choose 2 Box C/D RNAs (SR4 and SR29) and the 5S rRNA as RNA-seq indicated that their circular isoforms exist in the cells and they specifically interacted with *Pab*1020 RNA ligase in pull-down experiments. Fluorescent RNA substrates were prepared by *in vitro* transcription with T7 RNA polymerase capable of incorporating Cy5-labeled nucleotide analogs.

EMSA assays indicated that the 3 analyzed transcripts [Box C/D RNAs SR4 and SR29 (69mers) and the 5S rRNA (122mers)] formed specific RNA-protein complexes at near stoichiometric conditions (Fig. 3A). We also tested whether *Pab*1020 RNA ligase catalyzed the intramolecular ligation of the aforementioned transcripts (circularization) at 50°C. Fig. 3B shows that 15% acrylamide denaturing gel was not able to resolve the substrates and products of the RNA circularization reactions for these longer RNAs, as we successfully demonstrated in Fig. 1 for synthetic oligoribonucleotides. Therefore, to further identify circular RNA molecules, we used RNase R exoribonuclease treatment to discriminate between circular products and linear substrate RNAs. We observed that fluorescent transcript corresponding to *P. abyssi* 5S rRNA became partially resistant to RNase R treatment after incubation with *Pab*1020, whereas the linear substrate RNA was totally degraded (Fig. 3B). We further confirmed the RNA ligation activity of *Pab*1020 on the 5S rRNA and Box C/D RNAs SR4 or SR29 using inverse PCR. Briefly, the RNA ligation reactions were treated with RNase R, followed by the reverse transcription of each RNA (Fig. 3C). The outward facing (inverse) primers (when compared with the genomic sequence) were expected to amplify only circular templates, whereas only a small RT-product would be observed on a linear RNA template (Fig. 3C). For the same 3 selected RNAs (Box C/D RNAs SR4 orSR29 and 5S rRNA), after incubation with *Pab*1020 RNA ligase, we performed RT-PCR with the divergent primers described above. Indeed, we observed a full-length RT-PCR product (indicated by the asterisk in Fig. 3D) confirming RNA circularization by *Pab*1020. As negative control, in absence of RNA incubation with *Pab*1020 RNA ligase, the full-length amplification products corresponding to circular molecules (5S, SR4, SR29) were not observed (Fig. 3D).

These results from RNase R treatments and inverse PCR amplifications, confirmed that the RNA ligase encoded by *Pab*1020 gene is a (hyper)thermophilic protein that catalyzes the intramolecular ligation of RNA molecules.

Discussion

In archaea, 2 different major families of RNA ligases have been described. RtcB has been mainly implicated in ligation of single stranded tRNA halves with 2′-3′-cyclic phosphate and 5′-OH that occurs during splicing of pre-tRNAs. The Rnl3 ligase family, represented by Pab1020 studied here, uses similar mechanism as DNA ligases where 3′-OH reacts with 5′-phosphate to circularize RNA molecules. As the abundant presence of circular RNAs in all kingdoms of life, including hyperthermophilic Archaea, has only recently emerged, we investigated physiologic significance of *Pab*1020 RNA ligase activity previously

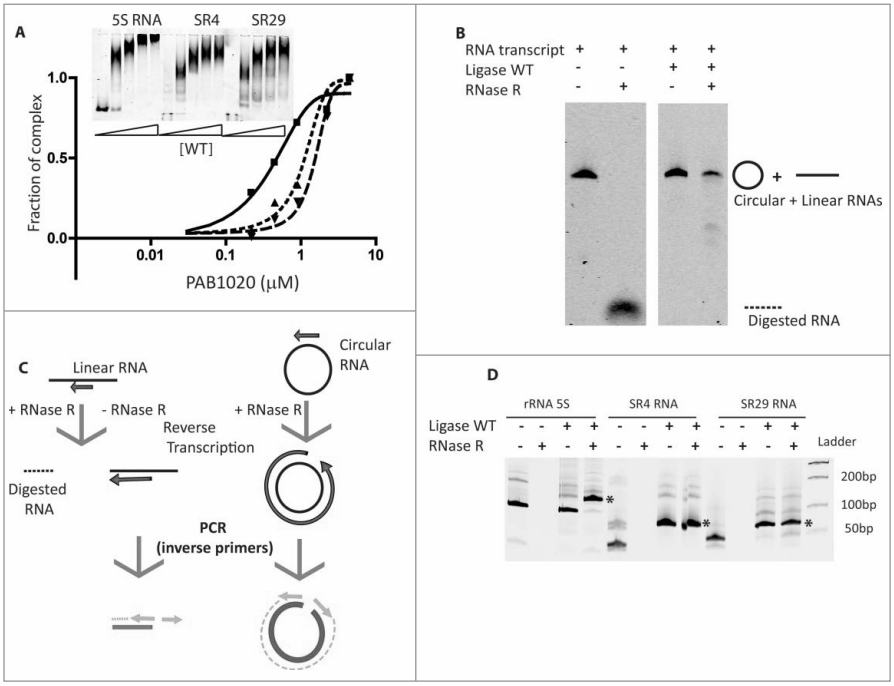


Figure 3. *Pab*1020 RNA ligase circularizes physiologically relevant RNA molecules. (A) RNA binding between *Pab*1020 RNA ligase (0.2 to 4.5 μM) and the *in vitro* transcripts (0.4 μM) corresponding to BoxC/D RNAs SR4 (■) and SR29 (▲) and 5S rRNA (▼) was analyzed by EMSA. A fraction of protein-RNA complex formed was plotted as a function of input protein. Insert: On the EMSA gel, the amount of the higher molecular weight bands, corresponding to *Pab*1020-nucleic acid complexes, increased as a function of the protein concentration. (B) *In vitro* transcript of 5S rRNA was incubated (right panel) or not (left panel) with *Pab*1020 RNA ligase (WT) for 120 min at 55°C. After incubation, recovered RNAs were treated or not with exoribonuclease RNase R for 120 min at 37°C before analysis on a 7% acrylamide 8M urea gel. (C) Schematic illustration of RT-PCR experiments on linear and circular RNAs with divergent primers to distinguish linear RNAs from circular RNAs products after incubation with *Pab*1020 RNA ligase. Only reverse transcription and PCR reactions on a circular RNA template will lead to the total amplification of the substrate sequence. (D) cDNA generated using outward facing primers on RNAs previously incubated (+) or not (−) with *Pab*1020 RNA ligase and in the presence (+) or absence (−) of RNase R were separated by gel electrophoresis. A full-length product attesting to amplification of circular RNA molecules, indicated by the asterisk, was observed for 5S rRNA (128 bp), Box C/D SR4 RNA (68 bp) and Box C/D SR29 RNA (66 bp). Circularization was observed only in the presence of *Pab*1020 RNA ligase.

observed only using synthetic substrates. The specific goal of our studies was to identify *bona fide* substrates of the Rnl3 family of RNA ligases. Using EMSA assays, we observed not only binding of *Pab*1020 to ssRNA (Fig. 1A), but to ssDNA as well. Nevertheless, under these experimental conditions, circularization activity was specific for oligoribonucleotides (Fig. 1B). Experiments shown in Fig. 1C also agree with the previous observations suggesting that "dimerization" and "C-terminal" domains of the Rnl3 family members are critical for intramolecular RNA circularization activity.[25]

We next established an experimental and computational pipeline to identify linear (substrates) and circular (products) RNA molecules that specifically interact with *Pab*1020 in cells. Toward this goal, we first isolated total RNA or *Pab*1020 interacting RNA molecules from *P. abyssi* cells. Obtained RNA samples were then reverse transcribed and sequenced using Ion Torrent technology. During this experimental protocol, RNase III fragmentation of circular molecules was necessary to allow ligation of adapters required for high-throughput sequencing. In the cases were this fragmentation occurs close to the junction and/or the reverse transcription does not proceed to the junction, reads originating from circular molecules would be erroneously classified as linear reads. However, although we are likely to underestimate the number of "circular" reads, the precise fragmentation site differs for individual molecules and it remains unlikely that all circular reads for a given locus would be missed in our computational analysis. Please note that our computational pipeline is ideally suited for analyses on prokaryotic data sets where RNA splicing is rare, as the frequent splicing would introduce gaps inside the matches.

The obtained sequencing reads were analyzed using the computational criteria described in Fig. 2A to identify the inverted matches, indicative of RNA circularization. At the first stage of our analysis, our experiments cumulatively supported a total of 30 000 distinct putative circularization junctions. For a given locus, the ratio between circular and linear reads varied substantially from approximately 5% up to 91%, indicating large *in vivo* heterogeneity in efficiency of RNA circularization process. This observation suggests that RNA circularization may be regulated and/or favored in cases where 5′ and 3′ extremities are brought together e.g., by structural constraints. The highest amount of circular reads (91% of all reads) was found for the intron of tRNA-Trp that carries the C/D box motif. In this case, 5′ and 3′ extremities are maintained in a close proximity by a bulge-helix-bulge structure already before ligation, thus likely favoring intramolecular ligation.

The Venn diagram shown in Fig. 4 illustrates the combined results from 3 individual experimental conditions of RNA-seq experiments that altogether revealed 133 circular loci (black numbers in Fig. 4). Interestingly, 127 of these circular loci (95%) were found in an RNA ligase pull-down fraction, suggesting that *Pab*1020 is necessary for RNA circularization in *P. abyssi* cells. We also stress that both linear and circular RNA molecules co-precipitated with the *Pab*1020 RNA ligase

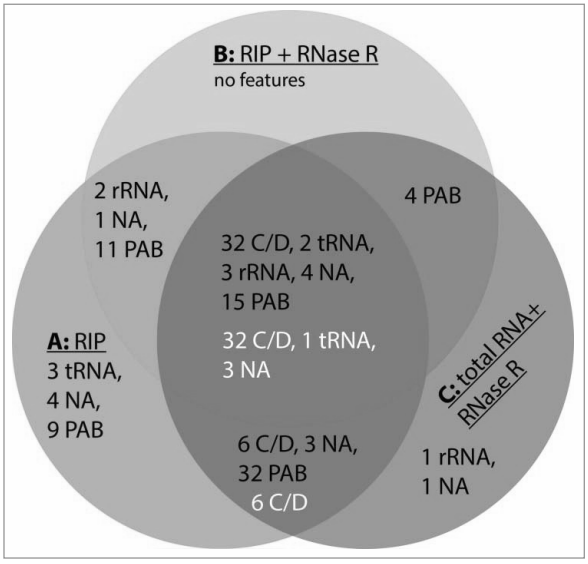


Figure 4. Venn diagram summarizing the results of our RNA-seq experiments. Samples used were: A, circular reads after RIP assays using *Pab*1020 antibodies; B, circular reads after RIP assay and ribonuclease R treatment; C, circular reads in total RNA samples treated with ribonuclease R. Black numbers refer to the categories of 133 junctions (Fig. 2B) and white numbers to 42 junctions with increased enrichment in RNase R experiments (Table 3).

(Table 2). Note that the intersection of the RIP assays (A), RIP assays with RNase R treatment (B) and total RNA with RNase R treatment degrading linear molecules (C) contained approximately 40% of all the circular RNA loci. In agreement with previous studies,[12,16] the most common circular RNAs correspond to the Box C/D RNA (guiding rRNA methylation) that showed an enrichment in RNase R experiments (indicated with white letters in Fig. 4, see also Table 3 for complete listing). These loci also suggested the presence of 3 novel non-coding RNAs that are evolutionary conserved within *Thermococcales*, indicating their functional importance. rRNAs, including *P. abyssi* 5S rRNA, and some coding RNAs did not show obvious enrichment in RNase R experiments, but have been observed in a circular form also in previous studies [Table S1.[10,30]]. Formation of the circular 5S rRNA could be compatible with the proposed processing mechanism of the archaeal pre-5S-RNA possibly[31] but it is unclear whether this potential circular form corresponds to an additional processing intermediate or the mature 5S rRNA.[15] When linear isoforms of the naturally occurring C/D box and 5 S rRNAs were used in binding (Fig. 3A) and circularization (Fig. 3B and C) assays, we observed an activity that was an order of magnitude higher than was observed for simple, likely non-structured, oligoribonucleotides (Fig. 1). As we have not observed intermolecular ligation in our assays (Figs. 1 and 3), we conclude that *Pab1020* is both necessary and sufficient for RNA circularization *in vitro* and *in vivo*. This notion is further supported by our observations indicating absence of tRNA splicing products in our pull-down reactions. Thus, 2 RNA ligase families are not interchangeable.

We postulate that RNA circularization could provide increased thermostability by limiting thermal denaturation of stem structures formed between the 5′ and 3′ termini of box C/D RNAs, which is in agreement with the preferential presence of Pab1020 orhtologs in many extremophiles. However, we stress that thermal protection of small non-coding RNAs does not alone explain the functional importance of RNA circularization, as *e.g.*, circular pseudouridylation guides carrying H/ACA motifs have not been reported to exist in archaea. The so called "H and ACA motifs" of these guides are obligatory found in single-stranded extremities of pseudouridylation guides,[32,33] thus likely disfavoring intramolecular ligation.

In conclusion, here we have presented the combined results from pull-down experiments, RNA-seq experiments and *in vitro* circularization assays revealing that *Pab*1020 is the key enzyme required for RNA circularization in Archaea. Our results suggest the duplication and functional speciation of an ancestral NTase domain and/or DNA ligase toward RNA ligase activity and prompt for further characterization of the widespread functional roles of circular RNAs in prokaryotes.

Materials and methods

Strains and cell culture techniques

P. abyssi GE5 was grown in continuous culture in a gas lift bioreactor as described previously.[34] Cells were collected in the exponential growth phase, followed by centrifugation at 6000 g for 15 min at 4°C. Strict anaerobic conditions were maintained during cell collection, centrifugation and storage of *P. abyssi* cells before further studies. Cell pellets were stored at −20°C.

Total RNA extraction from P. abyssi cells

Total RNA was isolated from approximately 10^8 *P. abyssi* cells following a single-step total RNA isolation protocol using the Tri-Reagent (Sigma-Aldrich). To remove contaminating DNA, 50 μg of isolated RNA was incubated with 50 units of RNase-free DNase I (New England Biolabs) for 30 min at 37°C. DNase I was inactivated by addition of 8 mM EDTA, pH 8 and 10 min incubation at 65°C. To obtain highly pure RNA samples, Tri-Reagent treatment was repeated to yield 40 μg of final RNA.

Production and affinity-purification of anti-Pab1020 antibodies

RNA ligase *Pab*1020 was produced and purified as described previously,[22] except that the Cobalt-Hi-Trap column was replaced by a nickel column. Different *Pab*1020 mutants constructed during this work have been detailed in supplementary Materials and Methods S1.

6 mg of purified *Pab*1020 RNA ligase were used to immunize 2 New Zealand rabbits (Genecust, Luxembourg). We used 5 mL Hi-Trap N-hydroxy-succinimide (NHS)-activated column (GE Healthcare) for the affinity purification of the antibodies following the procedure described in the GE Healthcare Antibody Purification Handbook (http://www.gelifesciences.com/handbooks). Briefly, the *Pab*1020 protein was linked to the active groups of the column and 6 mL antisera from immunized rabbits were passed through the column. Proteins bound non-specifically to the column were eliminated using several washing steps. Fractions containing *Pab*1020 specific antibodies were collected using acid elution, immediately neutralized, dialyzed against phosphate-buffered saline and concentrated to

1.5 mL. Specificity and titer of the obtained antibodies were confirmed by Western Immunoblots (Fig. S1B).

Formaldehyde cross-link and RNA immunoprecipitation (pull-down) assays

Approximately 10^{10} *P. abyssi* GE5 cells were suspended in 25 mM HEPES pH 7, 15 mM $MgCl_2$, 300 mM NaCl and were fixed using 2% formaldehyde during 20 min with gentle agitation. Crosslinking reactions were quenched using 100 mM glycine, followed by 2 successive washing steps in the same buffer as above. Obtained cell pellets were suspended in the extraction buffer containing 25 mM HEPES pH 7, 15 mM $MgCl_2$, 300 mM NaCl, 0.4 M Sorbitol and complete, EDTA-free Protease cocktail (Roche). To obtain a soluble fraction containing crosslinked RNA samples, at this stage precipitates were eliminated by a 10-minute centrifugation step at 14 000 g. The obtained soluble fractions contained approximately 30 ng/μl RNA (estimated using A_{260} values) and 0.1 mg.mL^{-1} protein determined using a Bradford protein assay.

For the RNA immunoprecipitation (RIP) assays, to reduce or eliminate non-specific binding, 300 μL of the above supernatant were incubated for 1 hour at 4°C with 20 μL Protein A-agarose (Sigma-Aldrich), followed by centrifugation for 2 min at 10 000 g at 4°C. The resulting supernatant was incubated at 4°C for 3 hours with 5 μL of purified rabbit Anti-*Pab*1020 antibodies before addition of 20 μL Protein A-agarose for an additional hour. The pellet was recovered after centrifugation for 2 min at 10 000 g. RNA-protein-complexes bound to the beads were washed 3 times with 25 mM HEPES pH 7, 15 mM $MgCl_2$, 300 mM NaCl and reversal of cross-links was achieved by incubation in the same buffer at 65°C for 1 hour. To recover RNA that specifically associated with the *Pab*1020 RNA ligase, samples were extracted with phenol/chloroform to remove proteins. For all samples, the remaining RNA was recovered using ethanol precipitation and dissolved in 20 μL water at a concentration of ~10 ng.μL^{-1}.

RNase R digestion of RNA samples

To enrich for circular RNA molecules, 100 ng of obtained RNA samples were treated with a magnesium dependent 3′ to 5′ exoribonuclease RNase R (Epicentre) at 37°C for 1 hour in a reaction buffer containing 20 mM Tris–HCl (pH 8), 0.1 mM $MgCl_2$ and 100 mM KCl. RNase R treatments were performed with a ratio of 1 unit of enzyme *per* 10 ng of RNA. Ethanol precipitation was performed to remove the enzymes and salts, followed by a second RNase R treatment. Exoribonuclease resistant circRNA molecules were extracted with phenol/chloroform, ethanol precipitated and suspended in water at ~10 ng.μL^{-1}.

Experimental circRNA-seq workflow

RNase R treated and non-treated RNA samples were sequenced using the Ion Total RNA-seq Kit V2 (Life Technology). Total and *Pab*1020 associated RNA samples were used for high throughput sequencing studies. Briefly, cDNA libraries were prepared for each sample containing 100 – 800 ng of RNA that was treated using RNase III that cleaves inter- or intramolecular regions of double-stranded RNA.[35] This resulted into formation of RNA fragments that were approximately 100-150 bp after 3 min incubation at 37°C with RNase III. RNA adaptor sequences were "splint ligated" to resulting linear RNA fragments using partly degenerate directional adapters. The first cDNA strand was reverse transcribed with the Superscript III Enzyme Mix (Life Technology) and double-stranded cDNA was amplified using Platinium PCR SuperMix High Fidelity (Life Technology) using manufacture's recommendations. Obtained DNA samples were diluted to obtain a final concentration of 100 pM, and were attached to beads and amplified using emulsion PCR. This circRNA protocol resulted into the clonal amplification of each RNA fragment within the microdroplets (Ion Spheres). Beads containing amplified DNA were enriched to eliminate empty spheres. Resulting samples were loaded onto an Ion 314 Chip V2 and sequenced in an Ion Personal Genome Machine Sequencer (PGM^{TM}, Life Technology). Polyclonal sequences originating from microbeads containing more than one template molecule were filtered out during automatic data processing using the dedicated IonTorrent server.

A computational pipeline for detection of putative circular reads

Sequencing reads were mapped to the *P. abyssi* GE5 reference genome (GenBank: NC_000868.1). Read mapping was performed using Blastn (version 2.2.26+)[36] with the Megablast option using the following default parameters: word size at least 11, gap opening penalty of 5, gap extending penalty of 2, mismatch penalty of 3 and match reward of 1. The default expectation value threshold of 10 was used, and the maximum number of outputs was limited to 250 alignments per query. The maximum number of allowed outputs was not limiting our analyses, as the highest observed number of alignments for any given query was 182. To detect putative circular reads in sequencing data, all reads having 2 matches (from the Blastn output) that together covered the whole read, were selected. We considered only the permuted matches (Fig. 2A), with no overlap on the reference genome that were located within a 10 000 nucleotide window on the genome sequence. When more than 2 nucleotides and less than 11 (our minimum word size parameter) were missing in a match to cover the read, we looked "naively" for the small complementary match. This data processing step resulted into 2 sequence alignment data files in bam format that corresponded to linear and putative circular reads, respectively.

Electrophoretic mobility shift assay (EMSA)

Internally labeled RNA and DNA oligonucleotides were used for EMSA assays. The oligonucleotides used were: AUUCCGAUAG(Cy5dT)GACUACA (RNA) and ATTCCGATAG(Cy5dT)GACTACA (see also Table S2). Where indicated, RNA or DNA oligonucleotides or *in vitro* transcripts (2.5 μM) were incubated with protein samples (0.5–10 μM) in gel shift buffer containing 10 mM Tris–HCl pH 7, 150 mM NaCl, 0.5 mM DTT, 2.5 mM $MgCl_2$, 0.01 mM ATP, 8 units of RNase Inhibitor (Biolabs). Binding reactions were performed at 50°C for 30 min

in a final volume of 20 μL, and were analyzed using native gel electrophoresis. The final concentration of loading buffer used was 2 mM Tris–HCl pH 7.5, 10% glycerol, 0.1 mM EDTA pH 8, 20 μg.mL^{-1} BSA, 0.1 mg.ml^{-1} Orange C. Samples were loaded onto 10% native acrylamide/bisacrylamide gel (19:1) and electrophoresed in TEG 1X buffer (40 mM Glycine, 0.5 mM EDTA pH 8, 250 mM Tris–HCl) at 100 V for 3 h at room temperature. Gels were visualized and analyzed using an Odyssey system (LI-COR) using the 700 nm channel.

Preparation of fluorescent transcripts using in vitro *transcription*

Fluorescent transcripts for box C/D RNAs SR4 and SR29 (Table S2) were transcribed *in vitro* using synthetic DNA oligonucleotides as DNA template (Eurogentec). DNA templates were double stranded in the region corresponding to the 17 nucleotides of the T7 promoter sequence (TAATACGACTCACTATA). These dsDNA templates were prepared by hybridizing oligonucleotides corresponding to 10 μM T7 Promoter (plus strand), 10 μM T7 Promoter-RNA gene (*Pab*snRNA32 or *Pab*snRNA35, minus strand) in T7 RNA polymerase buffer (2,5X), by 3 min at 80°C, followed by slow cooling to ambient temperature. For 5S rRNA transcript, transcription was performed using 1 μg of *Dra*I linearized pUC57 plasmid encoding T7 RNA polymerase promoter and 5S rRNA gene *Pab*r05.

Standard 20 μL transcription reactions contained dsDNA template (10 μM for box C/D RNA template or 0.1 μM for pUC57–5S RNA gene *Pab*r05), 7.5 mM of each NTP, 1X commercial reaction buffer, 12 units of RNasine, 0.25 mM Cy5-UTP and 2 μL Enzyme mix (T7 RNA Polymerase Megascript kit Ambion, Life Technology). All transcription reactions were allowed to proceed for 1 night at 37°C, before addition of 1 μL DNase Turbo (Megascript kit) and 15 min incubation at 37°C. Transcripts were analyzed using a 15% denaturing polyacrylamide gel electrophoresis (PAGE) and visualized by UV-shadowing. This allowed excision and elution of RNA from the gel with Maxam-Gilbert solution (0.5 M Na-acetate, 10 mM Mg-acetate, 1 mM EDTA, 0.1% SDS). RNA was precipitated using ethanol and transcripts were dissolved in diethylpyrocarbonate (DEPC) treated water to yield a final amount of approximately 700 pmol of transcript.

RNA circularization assays using fluorescent RNA and/or DNA oligonucleotides

Standard activity assays were performed in a mixture containing 10 mM Tris–HCl pH 7, 150 mM NaCl, 0.5 mM DTT, 2.5 mM $MgCl_2$, 8 units of RNase Inhibitor (Biolabs), 10 pmol Cy5-RNA or -DNA molecules and 200 pmol RNA ligase *Pab*1020 in a total volume of 20 μL. Reactions were initiated by adding enzymes and incubated for 90 min at 50°C. Proteins were extracted with phenol/chloroform and obtained RNA samples were suspended in 10 μL H_2O and 5μL denaturing buffer (Orange C 1 mg *per* mL in formamide). Reactions substrates and products were resolved using denaturing 18% PAGE containing 8M urea in 0.5X TBE. RNA molecules were revealed and quantified using an Odyssey imaging system as above (LI-COR).

RNA ligase assays on fluorescent box C/D and 5S RNA transcripts

Standard RNA ligase assays were performed as described above using Cy5-labeled box C/D or 5S transcripts. After incubation at 50°C for 120 min, RNA molecules were recovered by ethanol precipitation and resuspended in 10 μL of RNase R buffer 1X and incubated for 120 min at 37°C with 10 units of RNase R exoribonuclease (Epicentre), followed by 60 min incubation at 37°C with 40 μg of Proteinase K. Proteins were extracted by Phenol/chloroform treatment and RNA were recovered by ethanol precipitation and resuspended in 10 μL of water. RNase R resistant RNA molecules were detected and quantified using a 7% polyacrylamide gel containing 8M urea in 0.5X TBE. Reverse transcription and PCR (RT-PCR) with outward facing primers was also used to confirm RNA circularization. In this case, reaction products recovered either from RNase R treated or non-treated RNA ligation assays were reverse transcribed using M-MLV RT (50 units, Promega) and primers complementary to a central portion of box C/D RNA gene. cDNA templates were PCR amplified using Taq DNA polymerase and 2 divergent (outward facing) primers to anneal at the ends of the cDNA sequences. We performed 30 cycles of PCR and PCR products were visualized after electrophoresis on a 15% – 8 M urea polyacrylamide gels under denaturing conditions. Bands were visualized by ethidium bromide staining.

Disclosure of potential conflicts of interest

No potential conflicts of interest were disclosed.

Acknowledgements

The authors thank Joëlle Kuhn for production of Pab1020 antibodies. We are grateful to Ghislaine Henneke and Didier Flament for *P. abyssi* cells and thank Claire Toffano-Nioche, Marc Graille and Herman Van Tilbeurgh for many stimulating discussions and suggestions during this work. We also acknowledge Ursula Liebl for critical reading of the manuscript.

Funding

This work was supported by the Agence Nationale de la Recherche grant RETIDYNA. Work in our laboratory is supported by E. Polytechnique, CNRS and INSERM. Funding for open access charge: INSERM.

References

1. Ebbesen KK, Kjems J, Hansen TB. Circular RNAs: Identification, biogenesis and function. Biochim Biophys Acta 2015; 1859:163-8; PMID:26171810; http://dx.doi.org/10.1016/j.bbagrm.2015.07.007
2. Lasda E, Parker R. Circular RNAs: diversity of form and function. RNA 2015; 20:1829-42; PMID:25404635; http://dx.doi.org/10.1261/rna.047126.114
3. Vicens Q, Westhof E. Biogenesis of Circular RNAs. Cell 2014; 159:13-4; PMID:25259915; http://dx.doi.org/10.1016/j.cell.2014.09.005
4. Hansen TB, Jensen TI, Clausen BH, Bramsen JB, Finsen B, Damgaard CK, Kjems J. Natural RNA circles function as efficient microRNA sponges. Nature 2013; 495:384-8; PMID:23446346; http://dx.doi.org/10.1038/nature11993

5. Memczak S, Jens M, Elefsinioti A, Torti F, Krueger J, Rybak A, Maier L, Mackowiak SD, Gregersen LH, Munschauer M, et al. Circular RNAs are a large class of animal RNAs with regulatory potency. Nature 2013; 495:333-8; PMID:23446348; http://dx.doi.org/10.1038/nature11928
6. Salzman J, Chen RE, Olsen MN, Wang PL, Brown PO. Cell-type specific features of circular RNA expression. PLoS Genet 2013; 9: e1003777; PMID:24039610; http://dx.doi.org/10.1371/journal.pgen.1003777
7. Zhang Y, Zhang XO, Chen T, Xiang JF, Yin QF, Xing YH, Zhu S, Yang L, Chen LL. Circular intronic long noncoding RNAs. Mol Cell 2013; 51:792-806; PMID:24035497; http://dx.doi.org/10.1016/j.molcel.2013.08.017
8. Monat C, Cousineau B. Circularization pathway of a bacterial group II intron. Nucleic Acids Res 2016; 44:1845-53; PMID:26673697; http://doi.org/10.1093/nar/gkv1381
9. Murray HL, Mikheeva S, Coljee VW, Turczyk BM, Donahue WF, Bar-Shalom A, Jarrell KA. Excision of group II introns as circles. Mol Cell 2001; 8:201-11; PMID:11511373; http://dx.doi.org/10.1016/S1097-2765(01)00300-8
10. Danan M, Schwartz S, Edelheit S, Sorek R. Transcriptome-wide discovery of circular RNAs in Archaea. Nucleic Acids Res 2011; 40:3131-42; PMID:22140119; http://dx.doi.org/10.1093/nar/gkr1009
11. Salgia SR, Singh SK, Gurha P, Gupta R. Two reactions of Haloferax volcanii RNA splicing enzymes: joining of exons and circularization of introns. RNA 2003; 9:319-30; PMID:12592006; http://dx.doi.org/10.1261/rna.2118203
12. Singh SK, Gurha P, Tran EJ, Maxwell ES, Gupta R. Sequential 2′-O-methylation of archaeal pre-tRNATrp nucleotides is guided by the intron-encoded but trans-acting box C/D ribonucleoprotein of pre-tRNA. J Biol Chem 2004; 279:47661-71; PMID:15347671; http://dx.doi.org/10.1074/jbc.M408868200
13. Dalgaard JZ, Garrett RA. Protein-coding introns from the 23S rRNA-encoding gene form stable circles in the hyperthermophilic archaeon Pyrobaculum organotrophum. Gene 1992; 121:103-10; PMID:1427083; http://dx.doi.org/10.1016/0378-1119(92)90167-N
14. Lykke-Andersen J, Garrett RA. Structural characteristics of the stable RNA introns of archaeal hyperthermophiles and their splicing junctions. J Mol Biol 1994; 243:846-55; PMID:7966305; http://dx.doi.org/10.1006/jmbi.1994.1687
15. Tang TH, Rozhdestvensky TS, d'Orval BC, Bortolin ML, Huber H, Charpentier B, Branlant C, Bachellerie JP, Brosius J, Hüttenhofer A, et al. RNomics in Archaea reveals a further link between splicing of archaeal introns and rRNA processing. Nucleic Acids Res 2002; 30:921-30; PMID:11842103; http://dx.doi.org/10.1093/nar/30.4.921
16. Starostina NG, Marshburn S, Johnson LS, Eddy SR, Terns RM, Terns MP. Circular box C/D RNAs in Pyrococcus furiosus. Proc Natl Acad Sci U S A 2004; 101:14097-101; PMID:15375211; http://dx.doi.org/10.1073/pnas.0403520101
17. Watkins H, Bohnsack M. The box C/D and H/ACA snoRNPs: key players in the modification, processing and the dynamic folding of ribosomal RNA. Wiley Interdiscip Rev RNA 2012; 3:397-414; PMID:22065625; http://dx.doi.org/10.1002/wrna.117
18. Randau L. RNA processing in the minimal organism Nanoarchaeum equitans. Genome Biol 2012; 13:R63; PMID:22809431; http://dx.doi.org/10.1186/gb-2012-13-7-r63
19. Clouet d'Orval B, Bortolin ML, Gaspin C, Bachellerie JP. Box C/D RNA guides for the ribose methylation of archaeal tRNAs. The tRNATrp intron guides the formation of two ribose-methylated nucleosides in the mature tRNATrp. Nucleic Acids Res 2001; 29:4518-29; PMID:11713301; http://dx.doi.org/10.1093/nar/29.22.4518
20. Lykke-Andersen J, Aagaard C, Semionenkov M, Garrett RA. Archaeal introns: splicing, intercellular mobility and evolution. Trends Biochem Sci 1997; 22:326-31; PMID:9301331; http://dx.doi.org/10.1016/S0968-0004(97)01113-4
21. Englert M, Sheppard K, Aslanian A, Yates JR, 3rd, Soll D. Archaeal 3′-phosphate RNA splicing ligase characterization identifies the missing component in tRNA maturation. Proc Natl Acad Sci U S A 2011; 108:1290-5; PMID:21209330; http://dx.doi.org/10.1073/pnas.1018307108
22. Brooks MA, Meslet-Cladiere L, Graille M, Kuhn J, Blondeau K, Myllykallio H, van Tilbeurgh H. The structure of an archaeal homodimeric ligase which has RNA circularization activity. Protein Sci 2008; 17:1336-45; PMID:18511537; http://dx.doi.org/10.1110/ps.035493.108
23. Chambers CR, Patrick WM. Archaeal Nucleic Acid Ligases and Their Potential in Biotechnology. Archaea 2015; 2015:170571; PMID:26494982; http://dx.doi.org/10.1155/2015/170571
24. Gu H, Yoshinari S, Ghosh R, Ignatochkina AV, Gollnick PD, Murakami KS, Ho CK. Structural and mutational analysis of archaeal ATP-dependent RNA ligase identifies amino acids required for RNA binding and catalysis. Nucleic Acids Res 2016; 44:2337-47; PMID:26896806; http://dx.doi.org/10.1093/nar/gkw094
25. Torchia C, Takagi Y, Ho CK. Archaeal RNA ligase is a homodimeric protein that catalyzes intramolecular ligation of single-stranded RNA and DNA. Nucleic Acids Res 2008; 36:6218-27; PMID:18829718; http://dx.doi.org/10.1093/nar/gkn602
26. Zhelkovsky AM, McReynolds LA. Simple and efficient synthesis of 5′ pre-adenylated DNA using thermostable RNA ligase. Nucleic Acids Res 2011; 39:e117; PMID:21724605; http://dx.doi.org/10.1093/nar/gkr544
27. Zhelkovsky AM, McReynolds LA. Structure-function analysis of Methanobacterium thermoautotrophicum RNA ligase - engineering a thermostable ATP independent enzyme. BMC Mol Biol 2012; 13:24; PMID:22809063; http://dx.doi.org/10.1186/1471-2199-13-24
28. Unciuleac MC, Goldgur Y, Shuman S. Structure and two-metal mechanism of a eukaryal nick-sealing RNA ligase. Proc Natl Acad Sci U S A 2015; 112:13868-73; PMID:26512110; http://dx.doi.org/10.1073/pnas.1516536112
29. Marchler-Bauer A, Derbyshire MK, Gonzales NR, Lu S, Chitsaz F, Geer LY, Geer RC, He J, Gwadz M, Hurwitz DI, et al. CDD: NCBI's conserved domain database. Nucleic Acids Res 2015; 43:D222-6; PMID:25414356; http://dx.doi.org/10.1093/nar/gku1221
30. Doose G, Alexis M, Kirsch R, Findeiss S, Langenberger D, Machne R, Mörl M, Hoffmann S, Stadler PF. Mapping the RNA-Seq trash bin: unusual transcripts in prokaryotic transcriptome sequencing data. RNA Biol. 2013; 10:1204-10; PMID:23702463; http://dx.doi.org/10.4161/rna.24972
31. Holzle A, Fischer S, Heyer R, Schutz S, Zacharias M, Walther P, Allers T, Marchfelder A. Maturation of the 5S rRNA 5′ end is catalyzed in vitro by the endonuclease tRNase Z in the archaeon H. volcanii. RNA 2008; 14:928-37; PMID:18369184; http://dx.doi.org/10.1261/rna.933208
32. Balakin AG, Smith L, Fournier MJ. The RNA world of the nucleolus: two major families of small RNAs defined by different box elements with related functions. Cell 1996; 86:823-34; PMID:8797828; http://dx.doi.org/10.1016/S0092-8674(00)80156-7
33. Ganot P, Bortolin ML, Kiss T. Site-specific pseudouridine formation in preribosomal RNA is guided by small nucleolar RNAs. Cell 1997; 89:799-809; PMID:9182768; http://dx.doi.org/10.1016/S0092-8674(00)80263-9
34. Pluchon PF, Fouqueau T, Creze C, Laurent S, Briffotaux J, Hogrel G, Palud A, Henneke G, Godfroy A, Hausner W, et al. An extended network of genomic maintenance in the archaeon Pyrococcus abyssi highlights unexpected associations between eucaryotic homologs. PLoS One 2013; 8:e79707; PMID:24244547; http://dx.doi.org/10.1371/journal.pone.0079707
35. Nicholson AW. Ribonuclease III mechanisms of double-stranded RNA cleavage. Wiley Interdiscip Rev RNA 2014; 5:31-48; PMID:24124076; http://dx.doi.org/10.1002/wrna.1195
36. Camacho C, Coulouris G, Avagyan V, Ma N, Papadopoulos J, Bealer K, Madden TL. BLAST+: architecture and applications. BMC Bioinformatics 2009; 10:421; PMID:20003500; http://dx.doi.org/10.1186/1471-2105-10-421

References

[1] C R Woese and G E Fox. “Phylogenetic structure of the prokaryotic domain: the primary kingdoms.” In: *Proceedings of the National Academy of Sciences of the United States of America* 74.11 (1977), pp. 5088–90.

[2] Bruce Albers et al. *Molecular Biology of the Cell - 6th edition*. Garland Science, 2015.

[3] Yihwa Yang, Daniel T. Levick, and Caryn K. Just. “Halophilic, Thermophilic, and Psychrophilic Archaea: Cellular and Molecular Adaptations and Potential Applications”. In: *Journal of Young Investigators* (2007).

[4] Mark A Ragan. “Trees and networks before and after Darwin.” In: *Biology direct* 4 (2009), 43; discussion 43.

[5] P. Schattner, A. N. Brooks, and T. M. Lowe. “The tRNAscan-SE, snoscan and snoGPS web servers for the detection of tRNAs and snoRNAs”. In: *Nucleic Acids Research* 33.Web Server (2005), W686–W689.

[6] Arina D. Omer et al. “Homologs of Small Nucleolar RNAs in Archaea”. In: *Science* 288.5465 (2000).

[7] Ye Ding and Charles E Lawrence. “A statistical sampling algorithm for RNA secondary structure prediction.” In: *Nucleic acids research* 31.24 (2003), pp. 7280–301.

[8] Shilpa R Salgia, Sanjay K Singh, Priyatansh Gurha, and Ramesh Gupta. “Two reactions of Haloferax volcanii RNA splicing enzymes: joining of exons and circularization of introns.” In: *RNA (New York, N.Y.)* 9.3 (2003), pp. 319–30.

[9] M. Danan, S. Schwartz, S. Edelheit, and R. Sorek. “Transcriptome-wide discovery of circular RNAs in Archaea”. In: *Nucleic Acids Research* 40.7 (2012), pp. 3131–3142.

[10] Linda Szabo and Julia Salzman. “Detecting circular RNAs: bioinformatic and experimental challenges”. In: *Nat Rev Genet* 17.11 (Nov. 2016), pp. 679–692.

[11] Takayuki Horiuchi and Toshiro Aigaki. “Alternative trans-splicing: a novel mode of pre-mRNA processing”. In: *Biology of the Cell* 98.2 (2006), pp. 135–140.

[12] R Saldanha, G Mohr, M Belfort, and A M Lambowitz. “Group I and group II introns.” In: *FASEB journal : official publication of the Federation of American Societies for Experimental Biology* 7.1 (1993), pp. 15–24.

[13] “A map of human genome variation from population-scale sequencing”. In: *Nature* 467.7319 (Oct. 2010), pp. 1061–1073.

[14] The 1000 Genomes Project Consortium. “A global reference for human genetic variation”. In: *Nature* 526.7571 (Oct. 2015), pp. 68–74.

[15] *International Cancer Genome Consortium,* `http://icgc.org/`.

[16] *The Cancer Genome Atlas,* `http://cancergenome.nih.gov`.

[17] Ernest Jay, Robert Bambara, R. Padmanabhan, and Ray Wu. "DNA sequence analysis: a general, simple and rapid method for sequencing large oligodeoxyribonucleotide fragments by mapping". In: *Nucleic Acids Research* 1.3 (1974), pp. 331–354.

[18] Cliff Meldrum, Maria A Doyle, and Richard W Tothill. "Next-generation sequencing for cancer diagnostics: a practical perspective." In: *The Clinical biochemist. Reviews* 32.4 (2011), pp. 177–95.

[19] James M. Heather and Benjamin Chain. "The sequence of sequencers: The history of sequencing DNA". In: *Genomics* 107.1 (2016), pp. 1–8.

[20] F. Sanger and A.R. Coulson. "A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase". In: *Journal of Molecular Biology* 94.3 (1975), pp. 441–448.

[21] Allan M. Maxam and Walter Gilbert. "A new method for sequencing DNA". In: *PNAS* 74.2 (1977), pp. 550–564.

[22] Ruiqiang Li et al. "De novo assembly of human genomes with massively parallel short read sequencing." In: *Genome research* 20.2 (2010), pp. 265–72.

[23] Daniel R Zerbino and Ewan Birney. "Velvet: algorithms for de novo short read assembly using de Bruijn graphs." In: *Genome research* 18.5 (2008), pp. 821–9.

[24] EUGENE W. MYERS. "Toward Simplifying and Accurately Formulating Fragment Assembly". In: *Journal of Computational Biology* 2.2 (1995), pp. 275–290.

[25] P A Pevzner, H Tang, and M S Waterman. "An Eulerian path approach to DNA fragment assembly." In: *Proceedings of the National Academy of Sciences of the United States of America* 98.17 (2001), pp. 9748–53.

[26] Heng Li and Nils Homer. "A survey of sequence alignment algorithms for next-generation sequencing." In: *Briefings in bioinformatics* 11.5 (2010), pp. 473–83.

[27] Knut Reinert, Ben Langmead, David Weese, and Dirk J. Evers. "Alignment of Next-Generation Sequencing Reads". In: *Annual Review of Genomics and Human Genetics* 16.1 (2015), pp. 133–151.

[28] Z. Ning. "SSAHA: a fast search method for large DNA databases". In: *Genome Research* 11.10 (2001), 1725)1729.

[29] R. Li, Y. Li, K. Kristiansen, and J. Wang. "SOAP: short oligonucleotide alignment program". In: *Bioinformatics* 24.5 (2008), pp. 713–714.

[30] S.F. Altschul, W. Gish, W Miller, E.W. Myers, and D.J. Lipman. "The Design and Analysis of Computer Algorithms". In: *Journal of Molecular Biology* 215.3 (1990), pp. 403–410.

[31] H. Li and R. Durbin. "Fast and accurate short read alignment with Burrows-Wheeler transform". In: *Bioinformatics* 25.14 (2009), pp. 1754–1760.

[32] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg. "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome". In: *Genome Biology* 10.3 (2009), R25.

[33] Ben Langmead and Steven L Salzberg. "Fast gapped-read alignment with Bowtie 2". In: *Nat Meth* 9.4 (Apr. 2012), pp. 357–359.

[34] A. Dobin et al. "STAR: ultrafast universal RNA-seq aligner". In: *Bioinformatics* 29.1 (2013), pp. 15–21.

[35] P. Klus et al. "BarraCUDA - a fast short read sequence aligner using graphics processing units". In: *BMC Research Notes* 5.1 (2012), pp. 1–7.

[36] Ruibang Luo et al. "SOAP3-dp: Fast, Accurate and Sensitive GPU-Based Short Read Aligner". In: *PLoS ONE* 8.5 (2013). Ed. by Frederick C. C. Leung, e65632.

[37] Heng Li et al. "The Sequence Alignment/Map format and SAMtools." In: *Bioinformatics (Oxford, England)* 25.16 (2009), pp. 2078–9.

[38] Markus Hsi-Yang Fritz, Rasko Leinonen, Guy Cochrane, and Ewan Birney. "Efficient storage of high throughput DNA sequencing data using reference-based compression." In: *Genome research* 21.5 (2011), pp. 734–40.

[39] `https://genomicsandhealth.org/`.

[40] Dirk D Dolle et al. "Using reference-free compressed data structures to analyze sequencing reads from thousands of human genomes." In: *Genome research* 27.2 (2017), pp. 300–309.

[41] Xianwen Meng et al. "Circular RNA: an emerging key player in RNA world". In: *Briefings in Bioinformatics* (2016), bbw045.

[42] C Cocquerelle, B Mascrez, D Hétuin, and B Bailleul. "Mis-splicing yields circular RNA molecules." In: *FASEB journal : official publication of the Federation of American Societies for Experimental Biology* 7.1 (1993), pp. 155–60.

[43] W. R. Jeck et al. "Circular RNAs are abundant, conserved, and associated with ALU repeats". In: *RNA* 19.2 (2013), pp. 141–157.

[44] Sebastian Memczak et al. "Circular RNAs are a large class of animal RNAs with regulatory potency". In: *Nature* 495.7441 (2013), pp. 333–338.

[45] Julia Salzman, Charles Gawad, Peter Lincoln Wang, Norman Lacayo, and Patrick O. Brown. "Circular RNAs Are the Predominant Transcript Isoform from Hundreds of Human Genes in Diverse Cell Types". In: *PLOS ONE* 7.2 (Feb. 2012), pp. 1–12.

[46] Thomas B. Hansen et al. "Natural RNA circles function as efficient microRNA sponges". In: *Nature* 495.7441 (Mar. 2013), pp. 384–388.

[47] Reut Ashwal-Fluss et al. "circRNA Biogenesis Competes with Pre-mRNA Splicing". In: *Molecular Cell* 56.1 (2014), pp. 55–66.

[48] Nagarjuna Reddy Pamudurti et al. "Translation of CircRNAs". In: *Molecular Cell* 66.1 (2017), 9–21.e7.

[49] Julia Salzman, Raymond E. Chen, Mari N. Olsen, Peter L. Wang, and Patrick O. Brown. "Cell-Type Specific Features of Circular RNA Expression". In: *PLoS Genetics* 9.9 (2013). Ed. by John V. Moran, e1003777.

[50] William R Jeck and Norman E Sharpless. “Detecting and characterizing circular RNAs”. In: *Nature Biotechnology* 32.5 (2014), pp. 453–461.

[51] Erika Lasda and Roy Parker. “Circular RNAs: diversity of form and function”. In: *RNA* 20.12 (2014), pp. 1829–1842.

[52] Petar Glažar, Panagiotis Papavasileiou, and Nikolaus Rajewsky. “circBase: a database for circular RNAs”. In: *RNA* 20.11 (2014), pp. 1666–1670.

[53] Thomas B. Hansen, Morten T. Venø, Christian K. Damgaard, and Jørgen Kjems. “Comparison of circular RNA prediction tools”. In: *Nucleic Acids Research* 44.6 (2016), e58.

[54] K. Wang et al. “MapSplice: Accurate mapping of RNA-seq reads for splice junction discovery”. In: *Nucleic Acids Research* 38.18 (2010), e178–e178.

[55] Steve Hoffmann et al. “Fast Mapping of Short Sequences with Mismatches, Insertions and Deletions Using Index Structures”. In: *PLoS Computational Biology* 5.9 (2009). Ed. by David B. Searls, e1000502.

[56] Steve Hoffmann et al. “A multi-split mapping algorithm for circular RNA, splicing, trans-splicing and fusion detection”. In: *Genome Biology* 15.2 (2014), R34.

[57] Xiao-Ou Zhang et al. “Complementary Sequence-Mediated Exon Circularization”. In: *Cell* 159.1 (2014), pp. 134–147.

[58] Jakub O. Westholm et al. “Genome-wide Analysis of Drosophila Circular RNAs Reveals Their Structural and Sequence Properties and Age-Dependent Neural Accumulation”. In: *Cell Reports* 9.5 (2014), pp. 1966–1980.

[59] Yuan Gao, Jinfeng Wang, and Fangqing Zhao. “CIRI: an efficient and unbiased algorithm for de novo circular RNA identification”. In: *Genome Biology* 16.1 (2015), p. 4.

[60] J Lykke-Andersen, C Aagaard, M Semionenkov, and R A Garrett. “Archaeal introns: splicing, intercellular mobility and evolution.” In: *Trends in biochemical sciences* 22.9 (1997), pp. 326–31.

[61] Mark Adrian Brooks et al. “The structure of an archaeal homodimeric ligase which has RNA circularization activity.” In: *Protein science : a publication of the Protein Society* 17.8 (2008), pp. 1336–45.

[62] Hubert F. Becker et al. “High-throughput sequencing reveals circular substrates for an archaeal RNA ligase”. In: *RNA Biology* (2017), pp. 1–11.

[63] Allen W. Nicholson. “Ribonuclease III mechanisms of double-stranded RNA cleavage”. In: *Wiley Interdisciplinary Reviews: RNA* 5.1 (2014), pp. 31–48.

[64] Yongxian Yuan, Huaiqian Xu, and Ross Ka-Kit Leung. “An optimized protocol for generation and analysis of Ion Proton sequencing reads for RNA-Seq”. In: *BMC Genomics* 17.1 (2016), p. 403.

[65] Christiam Camacho et al. “BLAST+: architecture and applications.” In: *BMC bioinformatics* 10 (2009), p. 421.

[66] Todd M. Lowe. *Pyrococcus abyssi sRNA Genomic Loci,* `http://lowelab.ucsc.edu/snoRNAdb/Archaea/Pab-loci.html`.

[67] Thean Hock Tang et al. “RNomics in Archaea reveals a further link between splicing of archaeal introns and rRNA processing.” In: *Nucleic acids research* 30.4 (2002), pp. 921–30.

[68] Heng Li. *wgsim,* `https://github.com/lh3/wgsim`.

[69] Roxane Lestini et al. “Intracellular dynamics of archaeal FANCM homologue Hef in response to halted DNA replication.” In: *Nucleic acids research* 41.22 (2013), pp. 10358–70.

[70] S. K. Singh, P. Gurha, E. J. Tran, E. S. Maxwell, and R. Gupta. “Sequential 2’-O-Methylation of Archaeal Pre-tRNATrp Nucleotides Is Guided by the Intron-encoded but trans-Acting Box C/D Ribonucleoprotein of Pre-tRNA”. In: *Journal of Biological Chemistry* 279.46 (2004), pp. 47661–47671.

[71] Jean F Challacombe et al. “Complete genome sequence of Halorhodospira halophila SL1.” In: *Standards in genomic sciences* 8.2 (2013), pp. 206–14.

[72] Marie-Pierre Béal, Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. “Forbidden Words in Symbolic Dynamics”. In: *Advances in Applied Mathematics* 25.2 (2000), pp. 163–193.

[73] M. Crochemore, F. Mignosi, and A. Restivo. “Automata and forbidden words”. In: *Information Processing Letters* 67.3 (1998), pp. 111–117.

[74] Yannis Almirantis et al. “Optimal Computation of Avoided Words”. In: Springer, Cham, 2016, pp. 1–13.

[75] F. Mignosi, A. Restivo, and M. Sciortino. “Words and forbidden factors”. In: *Theoretical Computer Science* 273.1-2 (2002), pp. 99–117.

[76] Armando J. Pinho, Paulo J. S. G. Ferreira, and Sara P. Garcia. “On finding minimal absent words”. In: *BMC Bioinformatics* 11 (2009).

[77] Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. “Forbidden Factors and Fragment Assembly”. In: Springer, Berlin, Heidelberg, 2002, pp. 349–358.

[78] Gabriele Fici. “Minimal Forbidden Words and Applications”. Thèse. Université de Marne la Vallée, Feb. 2006.

[79] Greg Hampikian and Tim Andersen. “Absent sequences: nullomers and primes.” In: *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing* (2007), pp. 355–66.

[80] Claudia Acquisti, George Poste, David Curtiss, Sudhir Kumar, and Y Belosludtsev. “Nullomers: Really a Matter of Natural Selection?” In: *PLoS ONE* 2.10 (2007). Ed. by Steven Salzberg, e1022.

[81] Sara P. Garcia, Armando J. Pinho, João M. O. S. Rodrigues, Carlos A. C. Bastos, and Paulo J. S. G. Ferreira. “Minimal Absent Words in Prokaryotic and Eukaryotic Genomes”. In: *PLoS ONE* 6.1 (2011). Ed. by Christian Schönbach, e16065.

[82] Sara P. Garcia and Armando J. Pinho. “Minimal Absent Words in Four Human Genome Assemblies”. In: *PLoS ONE* 6.12 (2011). Ed. by Zhanjiang Liu, e29344.

[83] Raquel M. Silva, Diogo Pratas, Luísa Castro, Armando J. Pinho, and Paulo J. S. G. Ferreira. “Three minimal sequences found in Ebola virus genomes and absent from human DNA: Fig. 1.” In: *Bioinformatics* 31.15 (2015), pp. 2421–2425.

[84] Supaporn Chairungsee and Maxime Crochemore. “Using minimal absent words to build phylogeny”. In: *Theoretical Computer Science* 450 (2012), pp. 109–116.

[85] Maxime Crochemore, Gabriele Fici, Robert Mercaş, and Solon P. Pissis. “Linear-Time Sequence Comparison Using Minimal Absent Words & Applications”. In: Springer, Berlin, Heidelberg, 2016, pp. 334–346.

[86] Mohammad Saifur Rahman, Ali Alatabbi, Tanver Athar, Maxime Crochemore, and M. Sohel Rahman. “Absent words and the (dis)similarity analysis of DNA sequences: an experimental study”. In: *BMC Research Notes* 9.1 (2016), p. 186.

[87] Erik Aurell, Nicolas Innocenti, and Hai-Jun Zhou. “The bulk and the tail of minimal absent words in genome sequences”. In: *Physical Biology* 13.2 (2016), p. 026004.

[88] Davide Vergni et al. “Nullomers and High Order Nullomers in Genomic Sequences”. In: *PLOS ONE* 11.12 (2016). Ed. by Jürgen Brosius, e0164540.

[89] M.-P. Beal, M. Crochemore, and G. Fici. “Presentations of Constrained Systems With Unconstrained Positions”. In: *IEEE Transactions on Information Theory* 51.5 (2005), pp. 1891–1900.

[90] Gabriele Fici, Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. “Word assembly through minimal forbidden words”. In: *Theoretical Computer Science* 359.1-3 (2006), pp. 214–230.

[91] Takahiro Ota and Hiroyoshi Morita. “On a two-dimensional antidictionary construction using suffix tries”. In: *2015 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2015, pp. 2974–2978.

[92] M. Crochemore, F. Mignosi, A. Restivo, and S. Salemi. “Data compression using antidictionaries”. In: *Proceedings of the IEEE* 88.11 (2000), pp. 1756–1768.

[93] M. Crochemore and G. Navarro. “Improved antidictionary based compression”. In: *12th International Conference of the Chilean Computer Science Society, 2002. Proceedings.* IEEE Comput. Soc, pp. 7–13.

[94] Takahiro Ota and Hiroyoshi Morita. “On the On-line Arithmetic Coding Based on Antidictionaries with Linear Complexity”. In: *2007 IEEE International Symposium on Information Theory*. IEEE, 2007, pp. 86–90.

[95] Martin Fiala and Jan Holub. “DCA Using Suffix Arrays”. In: *Data Compression Conference (dcc 2008)*. IEEE, 2008, pp. 516–516.

[96] Takahiro Ota and Hiroyoshi Morita. “On the adaptive antidictionary code using minimal forbidden words with constant lengths”. In: *2010 International Symposium On Information Theory & Its Applications*. IEEE, 2010, pp. 72–77.

[97] Takahiro Ota and Hiroyoshi Morita. “On antidictionary coding based on compacted substring automaton”. In: *2013 IEEE International Symposium on Information Theory*. IEEE, 2013, pp. 1754–1758.

[98] T. Ota and H. Morita. “On a universal antidictionary coding for stationary ergodic sources with finite alphabet”. In: *ISITA*. IEEE, 2014, pp. 294–298.

[99] Carl Barton, Alice Heliou, Laurent Mouchard, and Solon P. Pissis. “Parallelising the Computation of Minimal Absent Words”. In: Springer, Cham, 2016, pp. 243–253.

[100] Alice Heliou, Solon P. Pissis, and Simon J. Puglisi. "emMAW: computing minimal absent words in external memory". In: *Bioinformatics* (2017).

[101] Peter Weiner. "Linear pattern matching algorithms". In: *Proceedings of the 14th Annual IEEE Symposium on Switching & Automata Theory (SWAT 14)*. Foundations of Computer Science. Los Alamitos, CA, USA: IEEE Computer Society, 1973, pp. 1–11.

[102] Edward M. McCreight. "A Space-Economical Suffix Tree Construction Algorithm". In: *Journal of the ACM* 23.2 (1976), pp. 262–272.

[103] Esko Ukkonen. "On-line construction of suffix trees". In: *Algorithmica* 14.3 (1995), pp. 249–260.

[104] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. "New indices for text: PAT Trees and PAT arrays". In: ed. by William B. Frakes and Ricardo Baeza-Yates. Prentice-Hall, Inc., 1992, pp. 66–82.

[105] Udi Manber and Gene Myers. "Suffix Arrays: A New Method for On-Line String Searches". In: *SIAM Journal on Computing* 22.5 (1993), pp. 935–948.

[106] M. Burrows, M. Burrows, and D. J. Wheeler. "A block-sorting lossless data compression algorithm". In: (1994).

[107] Paolo Ferragina and Giovanni Manzini. "Opportunistic data structures with applications". In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. FOCS '00. IEEE Computer Society, 2000, pp. 390–.

[108] Dan. Gusfield and Dan. *Algorithms on strings, trees, and sequences : computer science and computational biology*. Cambridge University Press, 1997, p. 534.

[109] Stefan Kurtz and Stefan. "Reducing the space requirement of suffix trees". In: *Software: Practice and Experience* 29.13 (1999), pp. 1149–1171.

[110] Juha Kärkkäinen and Peter Sanders. "Simple Linear Work Suffix Array Construction". In: *Proceedings of the 30th International Colloquium on Automata, Languages and Programming*. Ed. by Jos Baeten, Jan Lenstra, Joachim Parrow, and Gerhard Woeginger. Vol. 2719. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2003, pp. 943–955.

[111] Dong Kyue Kim, Jeong Seop Sim, Heejin Park, and Kunsoo Park. "Constructing suffix arrays in linear time". In: *Journal of Discrete Algorithms* 3.2 (2005), pp. 126–142.

[112] Pang Ko and Srinivas Aluru. "Space Efficient Linear Time Construction of Suffix Arrays". In: *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*. Ed. by Ricardo Baeza-Yates, Edgar Chávez, and Maxime Crochemore. Vol. 2676. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2003, pp. 200–210.

[113] Simon J. Puglisi, W. F. Smyth, and Andrew H. Turpin. "A taxonomy of suffix array construction algorithms". In: *ACM Computing Surveys* 39.2 (2007), pp. 1–31.

[114] Giovanni Manzini and Paolo Ferragina. "Engineering a Lightweight Suffix Array Construction Algorithm". In: *Algorithmica* 40.1 (2004), pp. 33–50.

[115] Ge Nong, Sen Zhang, and Wai Hong Chan. "Linear Suffix Array Construction by Almost Pure Induced-Sorting". In: *2009 Data Compression Conference*. IEEE, 2009, pp. 193–202.

[116] Yuta Mori. *An implementation of the induced sorting algorithm (SA-IS)*. 2008.

[117] M. Salson, T. Lecroq, M. Léonard, and L. Mouchard. "Dynamic extended suffix arrays". In: *Journal of Discrete Algorithms* 8.2 (2010), pp. 241–257.

[118] Jeffrey Scott Vitter and Jeffrey Scott. "Algorithms and Data Structures for External Memory". In: *Foundations and Trends® in Theoretical Computer Science* 2.4 (2006), pp. 305–474.

[119] Timo Bingmann, Johannes Fischer, and Vitaly Osipov. "Inducing Suffix and LCP Arrays in External Memory". In: *Journal of Experimental Algorithmics* 21.1 (2016), pp. 1–27.

[120] Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. "Parallel External Memory Suffix Sorting". In: Springer, Cham, 2015, pp. 329–342.

[121] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. "Replacing suffix trees with enhanced suffix arrays". In: *Journal of Discrete Algorithms* 2.1 (2004). The 9th International Symposium on String Processing and Information Retrieval (SPIRE), pp. 53–86.

[122] Juha Kärkkäinen and Dominik Kempa. "LCP Array Construction in External Memory". In: *Journal of Experimental Algorithmics* 21.1 (2016), pp. 1–22.

[123] David Clark. "Compact PAT trees". PhD thesis. University of Waterloo, Canada, 1996.

[124] J. Ian Munro. "Tables". In: Springer, Berlin, Heidelberg, 1996, pp. 37–42.

[125] Gonzalo Navarro and Eliana Providel. "Fast, Small, Simple Rank/Select on Bitmaps". In: *Proceedings of the 11th international conference on Experimental Algorithms*. Springer-Verlag, 2012, pp. 295–306.

[126] Peter J A Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. "The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants." In: *Nucleic acids research* 38.6 (2010), pp. 1767–71.

[127] H. Li. "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM". In: *Genomics* (2013).

[128] Hirotada Fukae, Takahiro Ota, and Hiroyoshi Morita. "On fast and memory-efficient construction of an antidictionary array". In: *2012 IEEE International Symposium on Information Theory Proceedings*. IEEE, 2012, pp. 1092–1096.

[129] Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. "Versatile Succinct Representations of the Bidirectional Burrows-Wheeler Transform". In: *ESA*. LNCS. Springer, 2013, pp. 133–144.

[130] Carl Barton, Alice Heliou, Laurent Mouchard, and Solon P Pissis. "Linear-time computation of minimal absent words using suffix array". In: *BMC Bioinformatics* 15.1 (2014), p. 388.

[131] G. Jacobson and G. "Space-efficient static trees and graphs". In: *30th Annual Symposium on Foundations of Computer Science*. IEEE, 1989, pp. 549–554.

[132] Takahiro Ota, Hirotada Fukae, and Hiroyoshi Morita. "Dynamic construction of an antidictionary with linear complexity". In: *Theor. Comput. Sci.* 526 (2014), pp. 108–119.

[133] Martin Senft. "Suffix Tree for a Sliding Window: An Overview". In: *WDS*. Matfyzpress, 2005, pp. 41–46.

[134] Yannis Almirantis et al. "On avoided words, absent words, and their application to biological sequence analysis". In: *Algorithms for Molecular Biology* 12.1 (2017), 5:1–5:12.

[135] Yuta Fujishige, Yuki Tsujimaru, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. "Computing DAWGs and Minimal Absent Words in Linear Time for Integer Alphabets". In: *MFCS*. Vol. 58. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 38:1–38:14.

[136] A. Heliou, M. Léonard, L. Mouchard, and M. Salson. "Efficient dynamic range minimum query". In: *Theoretical Computer Science* 656 (2016), pp. 108–117.

[137] Jozef Haleš, Alice Héliou, Ján Maňuch, Yann Ponty, and Ladislav Stacho. "Combinatorial RNA Design: Designability and Structure-Approximating Algorithm in Watson–Crick and Nussinov–Jacobson Energy Models". In: *Algorithmica* (2016), pp. 1–22.

université PARIS-SACLAY
ÉCOLE DOCTORALE
INTERFACES
Approches Interdisciplinaires : Fondements, Applications et Innovation

Titre : Analyse des séquences génomiques : Identification des ARNs circulaires et calcul de l'information négative

Mots clefs : Séquençage, ARN, ARN circulaire, algorithmique, recherche de motifs

Résumé : Le développement des techniques de séquençage à haut débit a permis de nombreuses avancées dans les domaines de la biologie et de la santé. Les données sont produites en grande quantité à des coûts toujours plus faibles, cependant leur stockage et leurs analyses demeurent de vastes sujets de recherche.
Dans un premier temps nous avons étudié l'identification des ARNs circulaires à partir des données de séquençage. L'alignement de ces données, appelées des *lectures*, pour identifier les ARNs circulaires est particulier. En effet, avant d'être séquencés les ARNs sont fragmentés aléatoirement en morceaux de taille environ 100. Ceux-ci sont ensuite lus lors du séquençage, on obtient ainsi les lectures. La jonction d'un ARN circulaire peut se retrouver à des positions aléatoires sur les lectures. Celles-ci s'alignent donc seulement partiellement à deux endroits sur le génome, au lieu d'avoir un match global. Nous avons proposé une nouvelle méthode permettant d'identifier les ARNs circulaires chez les Archées et les Bactéries. Nos résultats ont permis de montrer l'implication de la ligase de la famille Rnl3 dans la circularisation des ARNs chez l'archée *Pyroccoccus Abyssi*.
Dans un second temps, nous avons abordé de façon plus théorique l'analyse des séquences génomiques. L'analyse de ces séquences repose généralement sur leur alignement ou sur la distribution des mots présents. Nous nous sommes intéressés à une approche duale de celles-ci, en nous concentrant sur ce qui est absent, l'information négative. Plus précisément nous avons élaboré des algorithmes pour calculer les mots qui sont absents d'une séquence mais dont tous les facteurs sont présents, les *mots absents minimaux*. Nos algorithmes ont tous des complexités linéaires en temps et en espace, mais ils diffèrent sur le compromis entre temps de calcul et quantité de mémoire interne utilisée.

Title : Genomic sequences analysis: Identification of circular RNAs and computation of minimal absent words

Keywords : Sequencing, RNA, circular RNA, algorithms, pattern matching

Abstract : Improvements in high-throughput sequencing has enabled achievements in Biology and Health. Data are produced in large scale at low cost. However their storage and analysis remain problematic. In this PhD thesis, we tackled two problems related to NGS data analysis.
First we studied the identification of circular RNAs from sequencing data, called the *reads*. Alignment of reads coming from circular RNAs is special. Indeed before the sequencing step, the RNAs are randomly fragmented into pieces of size more or less 100. These are then sequenced into reads. The junction of a circular RNA can be located anywhere on the reads. Then these reads match only partially at two places instead of one global match. We proposed a novel pipeline to identify circular RNAs in Archaea and Bacteria. Our results evidenced the implication of the Rnl3 family's ligase in the circularization of RNAs in the Archaea *Pyroccoccus Abyssi*.
Secondly, we addressed a more theoretical problem of genomic sequences analysis. Usually these sequences are analysed on the basis of their alignment or of their words distribution. We focused on a dual approach, based on what is absent from the sequence, the negative information. More precisely we devised several algorithms to compute the *minimal absent words*. They are absent from the sequence but their factors are all present. Our algorithms have linear complexities in time and space but they vary in the trade-off between time and internal memory consumption.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France