Towards Dynamic Programming on Generalized Data Structures

and Applications of Dynamic Programming in Bioinformatics

Der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

Doctor rerum naturalium
(Dr. rer. nat.)

im Fachgebiet

Informatik

vorgelegt von

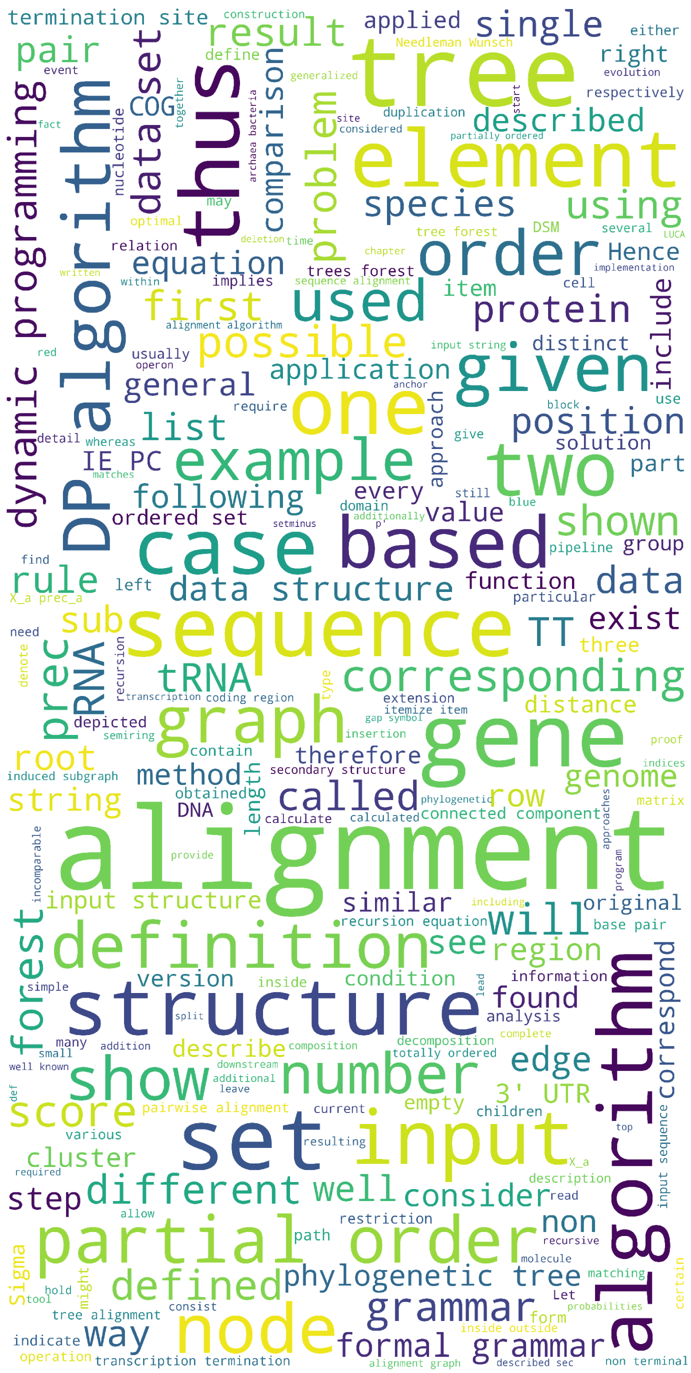
Master of Science Bioinformatik Sarah J. Berkemer
geboren am 23.03.1990 in Speyer

Die Annahme der Dissertation wurde empfohlen von:

1. Prof. Dr. Peter F. Stadler, Universität Leipzig, Deutschland

2. Prof. Dr. Danny Barash, Ben-Gurion Universität, Israel

Die Verleihung des akademischen Grades erfolgt mit Bestehen der Verteidigung am 05.02.2020 mit dem Gesamtprädikat *summa cum laude*.


alignment
tree
element
sequence
structure
algorithm
thus
DP
dynamic programming
case
based
gene
graph
definition
one
two
given
order
used
example
input
set
node
partial order
show
number
possible
result
single
species
using
protein
dataset
comparison
problem
first
equation
list
general
application
position
following
corresponding
tRNA
RNA
prec
sub
root
string
called
method
genome
row
score
different
well
consider
step
cluster
defined
phylogenetic tree
grammar
formal grammar
way
forest
region
found
edge
3' UTR
will
similar
original
see
version
condition
describe
empty
non
correspond
data structure
data
exist
rule
function
shown
value
every
approach
include
described
pair
termination site
applied
right
COG
Hence
item
distinct
solution
part
group
distance
therefore
three
TT
IE PC
ordered set
left
input structure
information
analysis
cluster
Sigma
path
restriction
transcription termination

Bibliographic Description

Title:	Towards Dynamic Programming on Generalized Data Structures
Subtitle:	and Applications of Dynamic Programming in Bioinformatics
Type:	Dissertation
Author:	Sarah J. Berkemer
Year:	2019
Professional discipline:	Computer Science
Language:	English
Key Words:	Dynamic Programming, Generalized Alignments, Data Structures, Applications of Dynamic Programming in Bioinformatics

This thesis is based on the following publications.

S. J. Berkemer, L.-K. Maier, F. Amman, S. H. Bernhart, J. Wörtz, P. Märkle, F. Pfeiffer, P. F. Stadler, and A. Marchfelder (2020). "Identification of RNA 3' ends and termination sites in Haloferax volcanii". In: *RNA Biology* 0.0, pp. 1–14. DOI: 10.1080/15476286.2020.1723328.

S. J. Berkemer, A. Hoffmann, C. R. Murray, and P. F. Stadler (2017a). "SMORE: Synteny Modulator of Repetitive Elements". In: *Life* 7.4, p. 42. DOI: 10.3390/life7040042.

S. J. Berkemer, C. Höner zu Siederdissen, and P. F. Stadler (2017b). "Algebraic dynamic programming on trees". In: *Algorithms* 10, p. 135. DOI: 10.3390/a10040135.

S. J. Berkemer, C. Höner zu Siederdissen, and P. F. Stadler (2019). "Compositional Properties of Alignments". In: *Mathematics in Computer Science, submitted.*

S. J. Berkemer and S. E. McGlynn (2020). "Phylogenetic domain separation of protein families constrains functional inference of LUCA". In: *Molecular Biology and Evolution, under review.*

C. A. Velandia-Huerto*, S. J. Berkemer*, A. Hoffmann, N. Retzlaff, L. C. Romero Marroquín, M. Hernández Rosales, P. F. Stadler, and C. I. Bermúdez-Santana (2016). "Orthologs, turn-over, and remolding of tRNAs in primates and fruit flies". In: *BMC Genomics* 17, p. 617. DOI: 10.1186/s12864-016-2927-4.

*The authors share first authorship.

Abstract

Dynamic programming (DP) is a way to solve optimization problems where the problem is divided into overlapping subproblems and the overall solution is composed out of optimal solutions to these subproblems. The solution to a subproblem is computed once and then stored in a table such that it can be reused when needed. In this way, a search space of exponential size can be explored in polynomial time. Dynamic programming was developed by Bellman in 1952, and early applications included detection of typing errors in programming code.

DP algorithms are widely used in bioinformatics, e.g., for the comparison of genomic sequences, called sequence alignment, or the prediction of molecule structures, also called folding. However, a steady increase of data sets and corresponding analytical methods requires more complex data structures to be used. Hence, the goal is to be able to apply dynamic programming to more than just strings. Within the framework of algebraic dynamic programming (ADP) it is possible to individually explore distinct building blocks of DP algorithms. This separation leads to the possibility of independently developing and changing such building blocks.

The thesis is divided into two parts where the first part describes theoretical aspects of DP algorithms and their development. It includes an introduction to dynamic programming (Chapter 2) which gives all relevant definitions and explanations for a better understanding of the subsequent chapters. The second part of the thesis lists possible applications of DP algorithms to biological data sets to show challenges, advantages and limitations thereof. An introductory chapter (Chapter 5) explains backgrounds of biological data and a short overview of existing algorithms.

The theoretical part of the thesis is concerned with the development of DP algorithms based on ADP and the generalization of input structures. A basic assumption of DP algorithms is the existence of a certain underlying order of elements inside the input structure which is the basis for a structure preserving traversal and decomposition of the input. In order to be able to apply DP algorithms to trees and forests, Chapter 3 defines decomposition operators which are then used to formulate algorithms such as alignment and editing on trees and forests. The algorithms are described by recursion equations as well as formal grammars within the framework of ADP. This allows to additionally formulate grammars for the application of inside-outside versions of the algorithms to calculate probabilities.

A large part of bioinformatics research is concerned with alignments. Here, the applications reach from simple comparisons of sequences to determine relations thereof or tracing back evolutionary history to finding mutations, detecting repetitive areas, or

mapping sequencing data to reference genomes. These applications are based on strings, however, one might want to compare more sophisticated structures such as trees or graphs, e.g., to compare molecules structures. Chapter 4 describes structural properties of alignments. Here, the definition of alignments on total orders (strings) is given and it is shown how alignments can be represented as alignment graphs. Based on this graph structure, the alignment definition is extended to partial orders and then further to alignments on general structures. Here, alignments of trees and forests as well as graphs are given as examples.

Another important aspect in bioinformatics research and application of algorithms is the possibility to adapt existing algorithms to new purposes. As data sets differ in various aspects, one wants to be able to use the basic principles of the algorithm but modify certain conditions. The second part of the thesis describes applications of DP algorithms including two modified versions of existing DP algorithms.

Chapter 6 describes a method to detect duplicated genetic elements. Here the alignment algorithm on strings is modified such that n:1 or 1:n matches can be detected. The algorithm is developed within the framework of ADP such that only a few parts had to be adapted and modified. The modified alignments are used to trace back the evolution of repetitive elements in primate genomes by detecting duplications within gene clusters.

Another DP algorithm on trees is the Fitch algorithm solving the minimal evolution problem. In Chapter 7, a modified version of the Fitch algorithm is used to find the minimal number of splits needed to separate archaeal and bacterial proteins inside a phylogenetic tree. Together with pairwise inter- and intra-domain distances, this leads to a measure of genes being well conserved during evolution or showing high amounts of horizontal gene transfer within the subtrees. One can see that this measure confirms earlier findings for the small ribosomal subunits that are assumed to being conserved since the last universal common ancestor (LUCA), thus, the last common ancestor of archaea and bacteria. Furthermore, it shows high variances for oxygen related groups of proteins confirming the assumption that oxygen concentration on early earth has been much lower than today.

Chapter 8 describes the analysis of sequencing data in order to detect transcription termination sites (TTS) in the archaeon *Haloferax volcanii*. Here, various algorithms are applied as a pipeline to find additional properties of those sites such as sequence or secondary structure motifs. In this way, TTS can be detected in a genome-wide approach which is the first of this kind and completely independent of existing gene annotations.

Zusammenfassung

Dynamische Programmierung (DP) ist eine Methode um Optimisierungsprobleme zu lösen. Hierbei wird das Problem in sich überlappende Teilprobleme unterteilt und eine optimale Lösung zu jedem der Teilprobleme berechnet. Diese werden dann wiederrum zur Gesamtlösung zusammengesetzt. Teillösungen werden in einer Tabelle gespeichert, sodass jede Teillösung nur einmal berechnet werden muss. So kann ein Suchraum exponentieller Größe in polynomieller Zeit durchsucht und eine optimale Lösung gefunden werden. Die dynamische Programmierung wurde 1952 von Bellman entwickelt und eine der ersten Anwendung war die Detektion von Tippfehlern beim Programmieren.

DP Algorithmen werden oft und sehr vielschichtig in der Bioinformatik angewendet wie zum Beispiel beim Vergleich von Gensequenzen, Sequenzalignment genannt, oder der Vorhersage von Molekülstrukturen. Die Menge an Daten und somit auch deren Analyse steigt stetig an, weshalb neue und komplexere Datenstrukturen immer wichtiger werden. Ein Ziel ist es deswegen, DP Algorithmen zu entwickeln, die auf komplexeren Datenstrukturen als Strings angewendet werden können. Durch das Prinzip der algebraischen dynamischen Programmierung (ADP) können DP Algorithmen in kleinere Bestandteile zerlegt werden, die dann unabhängig voneinander weiterentwickelt und abgeändert werden können.

Die Arbeit ist in zwei Teile gegliedert, wobei der erste Teil die theoretische Arbeit zur Entwicklung von Algorithmen der dynamischen Programmierung beinhaltet. Hierbei werden zuerst Prinzipien und Definitionen zur dynamischen Programmierung vorgestellt (Kapitel 2), um ein besseres Verständnis der darauffolgenden Kapitel zu gewährleisten. Der zweite Teil der Arbeit zeigt unterschiedliche bioinformatische Anwendungen von DP Algorithmen auf biologische Daten. In einem ersten Kapitel (Kapitel 5) werden Grundsätze biologischer Daten und Algorithmen vorgestellt, die dann in den weiteren Kapiteln benutzt werden.

Der theoretische Teil der Arbeit beschäftigt sich mit der Entwicklung von DP Algorithmen mit ADP und der Verallgemeinerung von Eingabestrukturen. DP Algorithmen beruhen auf der Annahme, dass die Elemente der Eingabestruktur basierend auf einer bestimmten Relation geordnet sind. Das bestimmt die Reihenfolge, in der die Eingabestruktur traversiert wird. Somit kann der DP Algorithmus auf schon berechnete kleinere Teillösungen zurückgreifen. In Kapitel 3 werden hierfür Operatoren definiert, die Baum- und Waldstrukturen in ihre Elemente zerlegen. Damit können dann DP Algorithmen definiert werden, wie Alignment und Editing auf Bäumen und Wäldern. Die Algorithmen werden durch Rekursionsgleichungen sowie formalen Grammatiken beschrieben. Basierend

auf der Beschreibung mit Grammatiken können dann außerdem Wahrscheinlichkeiten für das Alignment von Knoten in Bäumen berechnet werden.

In vielen Bereichen der bioinformatischen Forschung werden Sequenzalignments genutzt. Die Ziele sind hierbei oft sehr unterschiedlich und reichen vom einfachen Vergleich von Sequenzen um Verwandtheit oder evolutionäre Entwicklung zu bestimmen bis hin zur Identifikation von Mutationen, Detektion von repetetiven Regionen im Genom oder Mapping von Sequenzierdaten auf ein Referenzgenom. Diese Anwendungen basieren hauptsächlich auf Strings, wobei auch viele weitere Datenstrukturen in der Bioinformatik genutzt werden, wie zum Beispiel Bäume oder Graphen mit denen Molekülstrukturen dargestellt werden können. Kapitel 4 untersucht strukturelle Eigenschaften von Alignments und beginnt mit der Definition von Alignments auf total geordneten Strukturen (Strings). Alignments können auch als Graphen dargestellt werden. Basierend auf dieser Graphstruktur wird der Alignmentbegriff dann erweitert und auf partiell geordneten Strukturen definiert. Darüberhinaus werden Alignments dann auf allgemeinen Strukturen definiert. Als Beispiele werden Alignments auf Bäumen und allgemeinen Graphen gezeigt.

Ein weiterer wichtiger Punkt in der bioinformatischen Forschung und Anwendung von existierenden Algorithmen zur Analyse von Daten ist die Möglichkeit, Algorithmen auf eigene Daten und Bedürfnisse anzupassen. Dabei soll der Kern des Algorithmus oft bestehen bleiben während äußere Bedingungen angepasst werden. Zwei solche Fälle werden im zweiten und angewandten Teil der Arbeit vorgestellt.

Kapitel 6 beschreibt eine Methode um Genduplikationen zu detektieren. Hier wird der Alignmentalgorithmus auf Strings so angepasst, dass 1:n oder n:1 Matchings gefunden werden können. Dieser Algorithmus ist basierend auf ADP angepasst und wird angewandt um die evolutionäre Entwicklung von repetetiven Elementen in Primatengenomen zu erforschen.

Ein weiterer DP Algorithmus auf Baumstrukturen ist der Fitch Algorithmus, der basierend auf phylogenetischen Bäumen die minimale Anzahl an Mutationen während der Sequenzevolution von Genen berechnet. In Kapitel 7 wird eine modifizierte Version des Fitch Algorithmus angewandt, um die minimale Anzahl von Schnittkanten zu finden, die benötigt wird um Proteine aus Bakterien und Archeen in phylogenetischen Bäumen zu trennen. Zusammen mit einer weiteren Messung von Distanzen zwischen den Blättern der phylogenetischen Bäume kann man sehen, welche Proteine evolutionär gut konserviert sind und welche Proteine viele Mutationen und horizontalen Gentransfer aufzeigen. Wie durch andere Arbeiten in dem Feld bestätigt, zeigt sich, dass kleine Einheiten von ribosomalen Proteinen sehr gut konserviert sind. Dagegen zeigen Proteine, die am Sauerstoffmetabolismus beteiligt sind, sehr starke Veränderungen, was durch die Annahme bestätigt wird, dass die Sauerstoffkonzentration in frühen Stadien der Erdentwicklung geringer war als heute.

Kapitel 8 beschäftigt sich mit der Detektion von Transkriptionsterminationsstellen (TTS) im Archaeum *Haloferax volcanii*. Hier werden unterschiedliche Algorithmen angewandt um weitere Eigenschaften der Terminationsstellen zu finden, wie zum Beispiel Sequenz- oder Strukturmotive. Dadurch können TTS genomweit und unabhängig von existierenden Genannotationen gefunden werden.

Acknowledgment

Dear Peter, thank you very much for all the support! I very much enjoyed the time as a PhD student not only because I could do many different projects but also because I would always get a quick answer to all my questions. I did learn a lot about doing research but also about working in many different countries with a broad range of people from various (working) cultures. Thank you for giving a chance to so many people and for not stopping research at the door of the institute but making the world a research office.

Dear reviewers, thank you for reviewing this thesis even though it got a little bit longer than I planned.

Dear Petra, Jens, Sven, Andrea and Steve, you keep the system going! Thank you for keeping our Beerinformatics institute alive, being there to help everyone regardless of the problem size and secretly fixing things in order to create a great environment to do research!

Dear Berni and Fabian, thank you for so patiently teaching me how to work with sequencing data, answer all my questions and show me how to give extremely creative names to files containing the most important data sets of the project.

Dear colleagues at the Beerinformatics institute and also those who already left, thanks for all the good discussions, cakes, and coffee breaks! A special thanks to Fabian for creating the thesis template and for always quickly answering all the questions and demands for additional features!

I also would like to thank people from the MPI MIS and IMPRS in Leipzig, especially Antje Vandenberg, Jörg Lehnert and Jürgen Jost.

Dear Anita Marchfelder and Shawn McGlynn, thank you for the good collaboration and many discussions that taught me a lot - beside the research topics - about cooperations and communication in interdisciplinary research.

Dear coauthors and coworkers, I want to say thank you, as with every project, I learned a little bit more about research, collaborations, communication and organizing projects but also about other countries, disciplines and cultures.

Dear Marc, thank you for being so motivated when teaching, discussing and doing research! This is highly contagious!

Dear family and friends, dear Jugger team mates! Thank you for your support, endurance of my bad moods and a lot of good times together!

Dear Sven, thanks for always supporting and helping me in all areas, for so much time spent together, for the possibility to talk about everything and for giving me the permission to spend every summer in another country.

I hope to meet you all again at some time and some place in the world!

Contents

Part I

Introduction

CHAPTER 1

Dynamic Programming in Bioinformatics

Contents

Dynamic Programming was first described by Bellman (1952) and various algorithms are based on its principles (Sankoff, 1972; Sellers, 1974; Kruskal, 1983; Wagner and Fischer, 1974; Levenshtein, 1966). Dynamic programming was soon applied in the emerging fields of computer science to find typing errors in program code (Wagner and Fischer, 1974; Levenshtein, 1966), in computational biology to analyze and compare biological sequences (Needleman and Wunsch, 1970; Nussinov and Jacobson, 1980), and in linguistics to compare words of different languages (Kondrak, 2000; Cysouw and Jung, 2007; Steiner et al., 2011). Using dynamic programming techniques, search spaces of exponential size can be explored in polynomial time yielding an optimal solution to the stated optimization problem. Dynamic programming algorithms belong to the class of mathematical optimization problems and are closely related to Greedy and Divide-and-Conquer algorithms (Cormen et al., 2009).

The main principle of dynamic programming (DP) is the recursive traversal of the input structure, being decomposed into intermediate substructures, and the memoization (storage) of optimal results thereof. Hence, application of DP algorithms to for instance biological data leads to the question of how to best wrap the data into a feasible structure.

1.1 The Structure of Data Structures

The most basic data structures in computer science are sets and lists, together with cycles being closed lists and trivial cases such as the singleton element, the empty data structure and no structure at all. Algebraic definitions of data structures are unified by the theory of *combinatorial species*, developed by Joyal (1981). The concept of combinatorial species leads to the definition of combinatorial objects and corresponding generating functions. Irrespective of further aspects of combinatorial species, this introduction will make use of the basic definitions of data structures to give an intuition for their recursive construction and decomposition as well as their shapes and order of elements. Following Yorgey (2010) and Yorgey (2014), a set will be denoted by E, a list by L and a cycle by C. No structure at all is given by $\emptyset$, the empty structure by 1 and the singleton element is denoted X.

Data structures are defined and distinguished based on their internal structure, thus the way how elements inside the structure are set up. Given a set E, one can imagine a bag of elements which always has the same outer shape (a bag) independently of the number of contained elements. In contrast, a list L forms a sequence of elements where the size of the list grows with the number of its elements. The order of elements inside the list is fixed and removal or insertion of elements is restricted to its ends, as further described below. The structure of a cycle C is similar to a list with an additional connection of the first and last element. If a new element is inserted in the cycle, the order of the remaining elements is kept fixed. Elements in a data structure can only be distinguished by their order or if they are uniquely labeled. As there is no further internal structure in a set, elements of a set E can only be distinguished based on unique labels. Figure 1 depicts the basic data structures set, list and cycle with elements uniquely labeled by color and shape.

It is possible to recursively describe such data structures by the way they are constructed. To construct a list L, one starts with the empty structure 1 and increases its size by inserting a singleton element X using the corresponding concatenation operator $\bullet$. Thus,

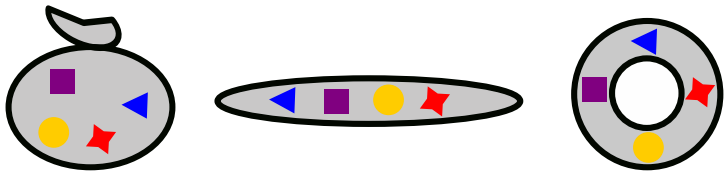


Figure 1: Basic data structures depicted as combinatorial species: set E (left), list L (middle) and cycle C (right) with four uniquely labeled elements.

a list is then algebraically formulated by

$$L = 1 + X \bullet L \tag{1.1}$$

with 1 being the empty list. The operator $+$ stands here for a logical 'or' ($\vee$) and the $\bullet$ operator stands for the concatenation of an element X to an existing list L. Thus, a list L is either empty (1) or contains at least one element concatenated to a list ($X \circ L$). The above description of data structures is based on the topic of combinatorial species coined by Joyal (1981). More detailed information can be found in Yorgey (2010), Yorgey et al. (2014), Bergeron et al. (1998), Yorgey (2014), and Flajolet and Sedgewick (2009).

1.2 Traversing Data Structures

Given a (labeled) data structure, a possible goal of an algorithm might be a structure preserving traversal such that all elements are visited. Given a list L, one way of traversal would be to start at one end and linearly traverse the elements until the other end, thus the same way the list has been constructed. In each step, the number of elements to be visited will decrease while the number of visited elements increases. In this way, each element will be visited once where the focused element is located at the border between visited elements and elements to be visited. In each step, the focus shifts towards the next element not yet visited. In a cycle C, one needs to identify a start element which then corresponds to the start element of a list resulting in an equivalent way of traversal. Given a set E of elements, there is no clear order or structure of elements that defines the way of traversal. Even with a distinguished start element, there are still many possible combinations of elements in the remaining set.

Such a traversal of data structures can be described by the concept of differentiation of combinatorial species. Algebraically, the differentiation defines the shape of surrounding data points with respect to a certain element. For combinatorial species, the derivative is a species with a 'hole', being the element in focus and thus, an empty data point in the current derivative (Yorgey, 2010; Yorgey, 2014; McBride, 2008). As described in Yorgey (2010) and Yorgey (2014), the derivative of the empty structure is no structure $1' \cong \emptyset$ and the derivation of the singleton element results in the empty structure $X' \cong 1$. As the shape of a set E does not change, its derivative is still a set, thus $E' \cong E$. A list decomposes into two lists around the 'hole' $L' = L^2$. Here, the lists represent the list of visited elements and the list of elements to be visited (McBride, 2008). A cycle

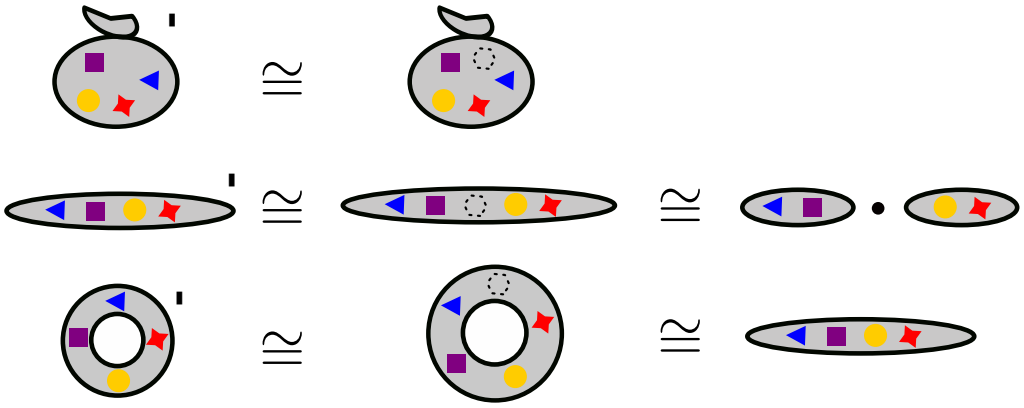


Figure 2: Basic data structure decomposition based on a the concept of species differentiation (Yorgey, 2010; Yorgey, 2014). The differentiation of species shapes the structure with respect to the focused element which becomes a 'hole' or empty element in the data structure. The empty element is depicted by the dotted circle. The structure of set E (top) is kept, the list L (middle) is decomposed into two lists around the 'hole' and the cycle C (bottom) is transformed into a list because the empty element splits its adjacent elements apart. The list decomposition can be thought of as the element in focus ('hole') with the elements on the right side being already traversed (yellow and red) and its left side still being to traversed (blue and purple) (McBride, 2008).

decomposes into a list at the location of the empty element, $C' = L$. Differentiation of E, L and C is depicted in Figure 2.

1.3 Dynamic Programming

A key element of a DP algorithm is the stepwise recursive traversal of the input structure in an order preserving way such that the same steps are applied to all substructures of the input. In the case of lists, the way of traversal corresponds to the recursive description of the data structure as shown in Equation 1.1 and thus, to the way of decomposing the data structure such that each element is visited as described by the differentiation of data structures in Figure 2 (Bird and De Moor, 1993; De Moor, 1994; McBride, 2008). Not every data structure can be described recursively or can easily be decomposed in a recursive manner. Hence, additional information such as a preset order of elements based on element indices is required to make sure that the data structure is traversed completely and in an order-preserving way. A *scoring function* or *scoring algebra* can be added to assign scores based on the labels or the structure of elements.

Classical formulations of DP algorithms are described by recursion equations. However, adaptation of recursion equations to more formal descriptions generalizes and extends the scope of underlying theory and applications. A possible formulation is based on formal grammars. Two main arguments support the more formal descriptions of DP algorithms such as formal grammars:

- Indices are implicitly included in the input structure. This is called index structure and usually corresponds to the way of traversal of the input. Thus, the formal description of the DP algorithm does not have to deal with indices.
- Algorithm formulation and scoring algebra are separated such that they can be developed independently and will later be connected.

The principle of separating building blocks of DP algorithms into independent entities has been developed within the framework of algebraic dynamic programming (ADP) (Giegerich, 2000; Giegerich and Meyer, 2002) which will be further explained in Subsection 2.2.7. More detailed definitions and descriptions of dynamic programming will be given in Chapter 2. Dynamic programming is based on sequential calculation of subsolutions. Thus, a (partially) ordered input structure with an initial (start) element is essential to solve the problem. The underlying index structure assures that smaller instances are calculated first and takes care for the correct storage and lookup of such intermediate solutions. This becomes important for the development of DP algorithms to more general data structures as described in Chapter 3 and Chapter 4.

1.4 Applied Dynamic Programming in Bioinformatics

In Bioinformatics, DP algorithms are a useful tool to solve tasks such as sequence alignment (Sankoff and Kruskal, 1983) or RNA secondary structure prediction (Hofacker et al., 1994; Lorenz et al., 2011a) (see Section 2.3). Here, the description of dynamic programming algorithms as formal grammars is common practice (Durbin et al., 1998).

Classical input structures to DP algorithms are strings or lists, or - in biological terms - *sequences*. Especially alignment algorithms used to compare two or more input structures make heavily use of strings as inputs. However, there is more than just sequences, and further data sets might require other data structures, e.g., trees and forests.

A simple, yet often encountered, example of a tree is the structure of a text such as a book. Books are divided into chapters, sections, subsections, and blocks of text, imposing a hierarchical structure in addition to the linear order of words. Many other formats for data exchange are based on similar structures, including `json` and `xml`. In bioinformatics, tree structures encode, among others, the (secondary) structure of RNA and phylogenetic trees as described in Subsection 2.3.2, hence DP algorithms on trees have a long history in computational biology. Well-studied problems include the computation of parsimony and likelihood scores for given phylogenies (Felsenstein, 1981), tree alignment (Chauve et al., 2016) and tree editing (Zhang and Shasha, 1989), reconciliation of phylogenetic trees (Jacox et al., 2016), and RNA secondary structure comparison (Schirmer et al., 2014; Rinaudo et al., 2012). Tree structures will be further explained in Subsection 2.2.1 and Chapter 3 from a computer science point of view and in Chapter 5 and Chapter 7 explaining biological background and applications.

Alignments play an important role in particular in bioinformatics as a means of comparing two or more strings by explicitly identifying correspondences between letters (usually called matches and mismatches) as well as insertions and deletions (Durbin et al., 1998). Alignments are calculated using dynamic programming where one of the first algorithms is the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970). The

aligned positions are interpreted either as deriving from a common ancestor ("homologous") or to be functionally equivalent. Alignments have also been explored as means of comparing words in natural languages, (see e.g. Kondrak (2000), Cysouw and Jung (2007), Steiner et al. (2011), and Bhattacharya et al. (2018)), as a convenient way of comparing ranked lists (Fagin et al., 2003), for comparison of text editions (Wolff, 2000; Tiepmar and G. Heyer, 2017), and to analyze synteny in the comparison of genomes (Grabherr et al., 2010; Velandia-Huerto et al., 2016).

Alignment problems are usually phrased as optimization problems where different cases obtain distinct scores optimizing towards a certain goal, e.g., minimizing or maximizing the sum of scores. Most commonly a scoring model is defined for pairs of sequences and generalized to multiple alignments as sums over certain pairwise alignments that are obtained as projections. The pairwise scoring is usually specified either in terms of matches or in terms of edit operations (insertions, deletions, or substitutions). Since alignments provide a method of specialized comparisons of two or more input structures, a desirable approach is to be able to apply the alignment algorithm to a large class of input structures. However, each input might still require distinct rules that are in general not needed. Hence, one wants the algorithm to still be adaptable to given constraints. This also holds for other DP algorithms, thus an extendable set of rules and scoring functions sets a good basis for applicable DP algorithms. A large part of the thesis will focus on alignments of generalized structures which includes strings and trees in Chapter 3 and further generalizations in Chapter 4. An extension of the alignment algorithm on strings is described in Chapter 6.

With the advent of sequencing technologies, biological sequence analysis became a main task in bioinformatics applications. Thus, sequence comparison algorithms are the underlying technique of many bioinformatics tools and appear in a large number of projects. In many cases, raw sequencing data is mapped to a reference genome as a first step in order to trace back the origins of sequence fragments. Given unknown fragments, a database search for similar sequences might reveal sequences of related organisms or genes. Sequence comparisons are also used to derive families of genes and their evolutionary history. An overview of popular algorithms and tools based on DP algorithms in bioinformatics is given in Section 5.4. Applications of the tools are described in Chapter 6, Chapter 7 and Chapter 8 which include applications and extensions of sequence comparisons as well as other algorithms based on dynamic programming.

1.5 Structure of the Thesis

This thesis will give an introduction to dynamic programming from the perspective of bioinformatics and corresponding applications. Therefore, it is composed out of two main parts. The first part describes theoretical aspects of dynamic programming including definitions and references to existing work in Chapter 2. The subsequent two chapters will deal with the stepwise generalization of input structures to DP algorithms, focusing on trees and forests as input structures in Chapter 3 and the creation of alignments on general data structures in Chapter 4. The chapters are based on Berkemer et al. (2017b) and Berkemer et al. (2019), respectively. The second part will deal with adaptations and extensions of DP algorithms and resulting applications to biological data with an introductory

chapter on the biological background and bioinformatics tools (Chapter 5). The next chapters then describe applications of DP algorithms reaching from own adaptations and implementations (Chapter 6, Chapter 7) to pure application (Chapter 8) based on Velandia-Huerto et al. (2016) and Berkemer et al. (2017a), Berkemer et al. (2020) and Berkemer and McGlynn (2020), respectively.

A concluding chapter (Chapter 9) will present an outlook on future work and perspectives on developments about theory and application of DP algorithms in bioinformatics.

Part II

Theoretical Aspects of Dynamic Programming

CHAPTER 2

Introduction to Dynamic Programming

Contents

Dynamic programming is a general paradigm for solving combinatorial problems with a search space of exponential size in polynomial time (and space). The key idea is that the optimal solution of the overall problem can be constructed by combining optimal solutions of subproblems. During the recursive deconstruction of the original problem instance, the same smaller subproblems are required repeatedly and thus memoized. Dynamic programming is typically applied to optimization and (weighted) enumeration problems. Not all such problems are amenable to this strategy however. Bellman's principle (Bellman, 1952) codifies the key necessary condition, the intuition which can be expressed for optimization problems as follows: The optimal solution of any (sub)problem can be obtained as a suitable combination of optimal solutions of its subproblems.

This chapter will start with a well-known example, the string-to-string correction problem (Section 2.1) to introduce dynamic programming. The main principles of a dynamic programming algorithm are (i) the recursive traversal and decomposition of its input and (ii) the memoization of intermediate results which will be explained in Subsection 2.1.1. Next to a formal definition of dynamic programming algorithms in Subsection 2.2.2, Section 2.2 will explain the relation to formal grammars (Subsection 2.2.3), to parsers (Subsection 2.2.4) and an algebraic way of writing dynamic programming algorithms, called algebraic dynamic programming (ADP) (Subsection 2.2.7).

The last part of this chapter will focus on two basic dynamic programming algorithms frequently used in bioinformatics applications. Basic principles of string alignments are described in Subsection 2.1.2 and a short review about the topic is given in Subsection 2.3.1. Subsection 2.3.2 describes the fundamentals of structure prediction based on dynamic programming. .

2.1 String-to-String Correction

The string-to-string correction problem was originally introduced to detect spelling errors (Glantz, 1956). Especially words that differ by exactly one letter from their original version as a result of a wrongly typed key were targeted and considered to be easily detectable and correctable (Morgan, 1970). The string-to-string correction problem will serve as an introductory example do dynamic programming (DP) algorithms. A formal definition will be given after this first example in Subsection 2.2.2.

As described by Navarro (2001), the string-to-string correction problem was introduced in various versions (Sankoff, 1972; Sellers, 1974; Kruskal, 1983; Wagner and Fischer, 1974; Levenshtein, 1966; Needleman and Wunsch, 1970) and applications reached from correction of spelling errors to comparisons of texts or biological sequences in evolutionary scenarios, summarized by the term string-to-string comparison which is an extension to string-to-string correction where algorithmic principles are very similar. Input to the string-to-string correction problem can be words of a natural language, key terms of a programming language, biological sequences or other word-like structures. Possible ways of depicting results of the string-to-string correction algorithm are shown in Figure 3. Here, one can clearly see diverging positions in the input strings and subsequences of characters that are in common.

Figure 3 shows three commonly used ways to depict results of string-to-string corrections (Kruskal, 1983). Figure 3 (left) shows a *trace* where matching positions of the first string

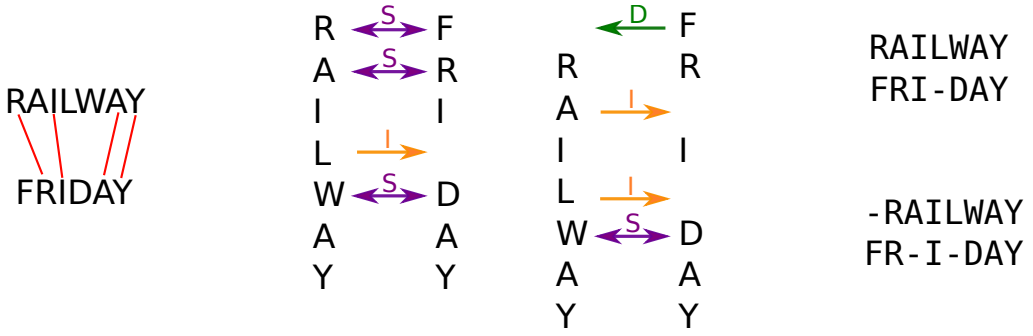


Figure 3: Comparison of the strings `railway` and `friday` depicted in three ways based on Kruskal (1983). The trace (left) shows corresponding letters in an order-preserving way by connecting them through non-crossing edges. Middle: Two listings of operations to transform the string `railway` into the string `friday`. Operations are substitution (S), deletion (D) and insertion (I). Both listings need a minimal number of operations (four) and thus, are co-optimals. Right: Alignments corresponding to the listings. Instead of listing the operations, input strings are depicted columnwise, whereas columns with a gap symbol '−' are insertions or deletions, columns with the same letters are matches and mismatches otherwise.

are connected to positions in the second string. Connections are required to be non-crossing and each element has at most one connection to an element in the other string. The middle part of Figure 3 shows two different *listings* of operations needed to transform `railway` into `friday`. This way of comparing strings is usually referred to as *string editing* based on edit operations such as insertion, deletion and substitution. The rightmost part of Figure 3 shows two alignments of the input strings. Each alignment corresponds to one of the listings described before. Instead of lists of edit operations, *string alignments* are depicted by a matrix with matrix columns showing matching positions of the input strings whereas a position in one string is matched to at most one position in another string. These are called alignment columns. If letters of a column are equal, they indicate a match case, if not, it is called a mismatch. Columns containing a *gap symbol* (−) indicate deletions or insertions whereas gap-only columns are not allowed. A formal definition of string alignment will be given in Subsection 2.1.2.

2.1.1 Calculating Distances between Strings

String-to-string correction is an optimization problem which aims to find the minimal number of operations needed to transform one string into another string. Basic edit operations are hereby substitution, deletion and insertion as shown in Figure 3. Based on the symmetric character of deletions and insertions, they are often summarized by the term *indel* (Kruskal, 1983).

Optimization problems are usually equipped with a scoring scheme in order to optimize towards a certain goal for instance reaching a minimal or maximal score. A basic scoring system for the comparison of two strings of equal length is the *Hamming distance* (Hamming, 1950). Thus for two strings $x = x_1 \dots x_n$ and $y = y_1 \dots y_n$ the Hamming

distance $d_H(x, y)$ is defined as the amount of positions in x and y that differ:

$$d_H(x, y) = \sum_{x_i \neq y_i} 1, \quad \text{for } i \in \{1, \dots n\} \tag{2.1}$$

For general strings (of various lengths), string editing is used which allows shifts in positions in order to maximize the number of matching letters. The optimality criterion is then a minimal number of edit operations where each operation is weighted by a score of 1, respectively. The overall score is the sum of weights of all operations needed to solve the problem. The listings in Figure 3(middle) differ, however, they both consist of a minimal number of operations. They are both optimal solutions to the string-to-string correction problem, also called co-optimals. This scoring model is called *Levenshtein distance* (Levenshtein, 1966). In contrast to the Hamming distance, the Levenshtein distance is defined on strings of distinct lengths as positional shifts are allowed when comparing strings $x = x_1 \dots x_n$ and $y = y_1 \dots y_m$ with:

$$d_L(i, j) = \min \begin{cases} d_L(i-1, j-1) & \text{if } x_i = y_j \\ d_L(i-1, j-1) + 1 & \text{if } x_i \neq y_j \\ d_L(i-1, j) + 1 & \\ d_L(i, j-1) + 1 & \end{cases}, \text{for } i \in \{1, \dots n\},\ j \in \{1, \dots m\}. \tag{2.2}$$

The Levenshtein distance is a widely used distance measure for string editing. The cases described in Equation 2.2 correspond to edit operations where the first case describes matching positions, the second case stands for substitutions and the last two cases show insertion and deletion. The costs for each operation except the match case are set to 1 (as the match case is not an edit operation, it does not 'cost' anything). It holds that the value of the Levenshtein distance is at most equal to the length of the longer input string. For strings of equal lengths, the *Hamming distance* (Hamming, 1950) is always an upper bound to the Levenshtein distance. The Hamming distance and the Levenshtein distance are 0 if and only if the input strings are equal.

The order of edit operations in the listing is of importance for the course of the algorithm being able to edit the same position in a string several times. This makes the algorithm quite flexible, however, also more time and space consuming than other variants of string comparison (Kruskal, 1983).

There exist many other scoring models (Navarro, 2001), e.g., the *edit distance* proposed by Wagner and Fischer (1974) where distinct edit operations have different costs which results in the algorithm preferring certain cases over others. Setting a higher weight for substitutions than for two indel operations, i.e. $w_{substitution} > 2w_{indel}$, it is always cheaper to only use indel and match cases. In this case, the algorithm is the same as the one for finding the largest common subsequence (Kruskal, 1983) and correlates with the trace depicted in Figure 3 by keeping matching positions untouched.

The *alignment distance* (Needleman and Wunsch, 1970) is a further extension to the edit distance where scores for substitutions are additionally differentiated by considering equality of letters in a column. In contrast to the distances explained above, matching position are given a score unequal to 0 which further extends the way the algorithm chooses the optimal solution.

2.1.2 Pairwise Comparison of Strings

As shown in Figure 3, there exist various versions of depicting commonalities and differences in two strings. In comparison to the trace showing matching letters or the listing of edit operations, alignments use gap characters ($-$) to indicate insertions and deletions. There are various ways of depicting alignments where a popular way is using a matrix representation.

Definition 1 (Alignment, *cf.* Chapter 4)
An alignment of k sequences $S = s^1, \ldots s^k$ of various lengths is a matrix with k rows corresponding to the sequences s^1 to s^k. Let s_i^a be the i-th position in sequence a, $a \in 1, \ldots k$ and $|s^a|$ denote the length of sequence a. The most common representation of an alignment is as a rectangular matrix whose k rows are indexed by the sequences and whose columns are indexed by integers $i \in [1, L]$, where L is the number of alignment columns. Each sequence $s \in S$ is then associated with a strictly monotonically increasing function $\alpha_s : [1, |s|] \to [1, L]$ such that for each $i \in [1, |s|]$, $\alpha_s(i)$ is the index of the column containing s_i. The alignment matrix contains a gap symbol $-$ in row s and column l whenever $\alpha_s^{-1}(l) = \emptyset$, otherwise, the matrix element is $s_{\alpha_s^{-1}(l)}$. Gap-only columns are excluded from the alignment.

Consider the following simple example:

```
a   001111100
b   0011011--
c   --1000100
i   123456789
```
(2.3)

The rows correspond to sequences a, b and c and row i indicates the indices of alignment columns. We have $\alpha_A(i) = i$ for $1 \leq i \leq 9$, $\alpha_B(i) = i$ for $1 \leq i \leq 7$, and $\alpha_C(i) = i + 2$ for $1 \leq i \leq 7$. For the 8th column of the example we have $\alpha_a^{-1}(8) = 8$, $\alpha_b^{-1}(8) = \emptyset$, $\alpha_c^{-1}(8) = 6$; hence the entries in the 8th column are $a_{\alpha_a^{-1}(8)} = a_8 = \texttt{0}$, $-$ because $\alpha_b^{-1}(8) = \emptyset$, and $c_{\alpha_c^{-1}(8)} = c_6 = \texttt{0}$. The actual values of the sequence elements, i.e., the s_i are of course important to determine the scoring.

A proper alignment is preserving the structure of its input, thus removing the gap symbols from the alignment rows will result in the original input sequences.

Definition 2 (Structure Preserving Alignment, *cf.* Chapters 3 and 4)
A string alignment is said to be structure preserving if there exists a map $\pi_a : [1, L] \to [1, |s^a|]$ with $s^a \in S$ and $a \in \{1, \ldots k\}$.

Figure 4 shows an example for the function π in the case of string alignment. More complex cases will be described in Chapter 3 on tree structures and in Chapter 4 for general structures.

Alignments are mainly used for string comparison, thus one can calculate a distance or similarity value of two strings whereas the optimality criteria are either minimizing the distance or maximizing the similarity score. Here, scores of alignment columns are summed to retrieve the overall score. Alignment columns correspond to cases match, mismatch, insertion and deletion that are scored independently (Needleman and Wunsch,

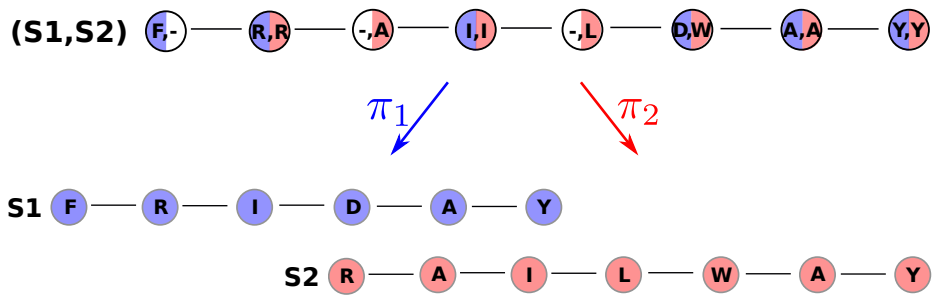


Figure 4: Example of the function π mapping the components of the aligned structure to its original structures, S1 and S2. Colors and positions of elements in (S1,S2) indicate from where they originated.

1970). Additionally, each position in the input can be changed at most once which makes the string alignment algorithm less flexible than string editing. Deleting the gap symbols from each sequence in the alignment will retrieve the original input sequence. Hence, an alignment of strings is the smallest common superstring of the inputs. This can be generalized to other input structures and is described in Chapter 4.

String editing and string alignment algorithms as described above are dynamic programming (DP) algorithms and commonly described by recursion equations. Thus, the same steps are applied to all instances of the problem, namely to all pairs of positions in the input strings. Recursion steps are indicated by repeated application of the function on a slightly changed input as also depicted in the recursion equations for the calculation of the Levenshtein distance in Equation 2.2. Here, the input to d_L are indices pointing to position in the input strings.

The following recursion equations (Equation 2.6) describe the string alignment problem optimizing for the smallest distance between the two input strings as, among others, described by Wagner and Fischer (1974) and Needleman and Wunsch (1970). The problem of string comparison is well-known in computer science and bioinformatics and described in various publications and textbooks, e.g. Durbin et al. (1998) and Cormen et al. (2009).

The inputs are two strings (also called sequences) $x = x_1x_2 \dots x_n$ and $y = y_1y_2 \dots y_m$. The operations are align, insert and delete with corresponding scores w_{align}, w_{insert} and w_{delete}. The recursion equations are based on indices i and j of the input strings with $1 \leq i \leq n$ and $1 \leq j \leq m$. In order to not (re)calculate the solutions of smaller problem instances again and again, results are stored in a matrix $d \in \mathbb{Z}^{n+1 \times m+1}$ which is called *memoization table*. Here, indices of a cell in the matrix match with the indices of the input strings. In this way, the recursion step draws the results for the smaller preceding instances from neighboring cells in the matrix as depicted in Figure 5 and specified in the recursion equations based on indices. The base case of the recursion is $d_{0,0} = 0$. The initialization of the algorithm calculates the scores for aligning each input sequence against

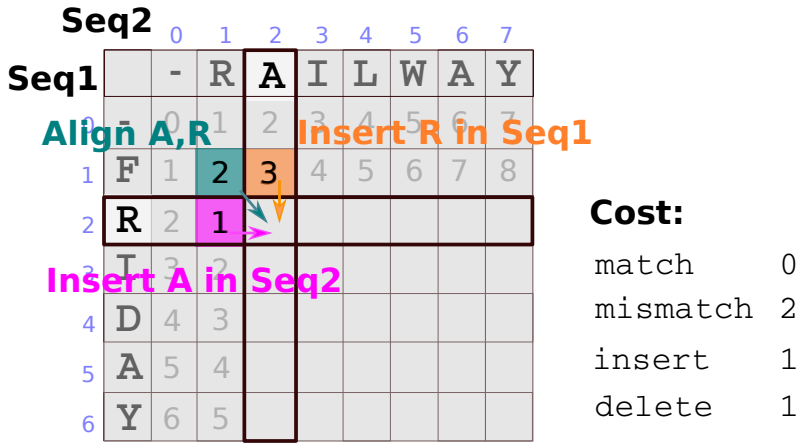


Figure 5: String-to-string comparison of the sequences `friday` (Seq1) and `railway` (Seq2) with operations align (green), insert (pink) and delete (orange) which is the same as inserting in the second input sequence. The cost function is written on the r.h.s where match and mismatch scores correspond to the alignment case for equal or unequal letters, respectively. The memoization table is filled such that small instances of the problem (= shorter strings) are calculated first. Then, current tiles can be filled by looking up values that have been calculated before. For each case, a specific cost and the value of the corresponding preceding tile are summed up. The resulting value in the current tile is the minimum of all (three) possibilities. Blue numbers indicate indices of the matrix and the corresponding input sequences. The complete solution is shown in Figure 6.

an empty string:

$$d_{0,j} = d_{0,j-1} + w_{delete}(-, y_j),\ 1 \leq j \leq m \tag{2.4}$$

$$d_{i,0} = d_{i-1,0} + w_{insert}(x_i, -),\ 1 \leq i \leq n \tag{2.5}$$

The optimized solution is the smallest possible value as indicated by the following recursion equations:

$$d_{i,j} = min \begin{cases} d_{i-1,j-1} + w_{align}(x_i, y_j), \\ d_{i-1,j} + w_{insert}(x_i, -), \\ d_{i,j-1} + w_{delete}(-, y_j). \end{cases} \tag{2.6}$$

The recursion equations describe how to calculate the result for every pair of indices (i, j) of the input strings. The result is the minimal value of three possible scores corresponding to the cases align (or substitute), insert and delete. They are composed out of a scoring value (w_{align}, w_{insert} or w_{delete}) and the result of the algorithm applied to a smaller instance of the problem (recursion step), shown by a decrement in at least one of the indices, thus $i-1$ or $j-1$. The recursion equations describe the top-down approach of the algorithm which starts from the complete sequence as an input. In contrast, the matrix is filled by a bottom-up approach, thus one starts from the empty sequences to fill the matrix (upper left corner) until the complete sequences are traversed (lower right corner of the matrix).

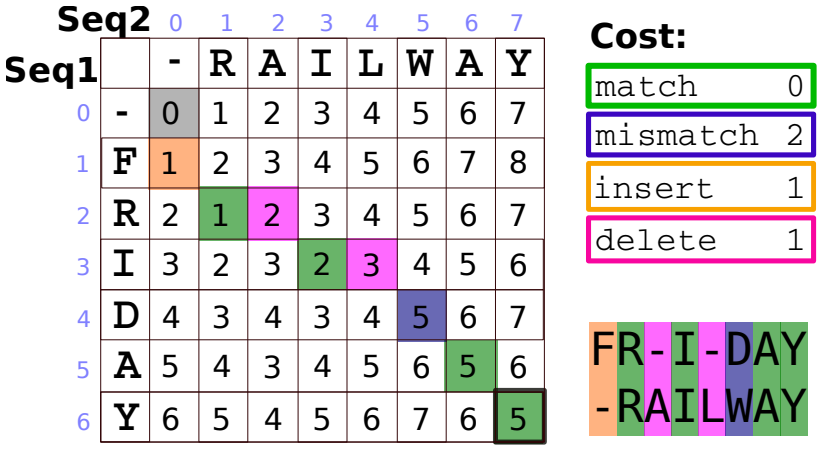


Figure 6: Complete memoization table for the string-to-string comparison of strings (or sequences) `friday` (Seq1) and `railway` (Seq2). Blue numbers indicate indices of the memoization table and input sequences. Scores written in cell (i, j) are the subsolution for substrings $(1, i)$ and $(1, j)$. Costs of rules are written on the r.h.s. with the final alignment below. The colored tiles show the backtracking steps corresponding to the cases match (green), mismatch (blue), insertion (orange) and deletion (pink). The final cost for the complete alignment is written in cell $(6, 7)$ and depicts the minimal distance between the input strings.

The scoring functions are applied on the current positions of the input strings, whereas w_{insert} and w_{delete} score an alignment column with a gap symbol. The score for the alignment case depends on the letters of the input:

$$w_{align}(x_i, y_j) = \begin{cases} \texttt{match} & \text{if } x_i = y_j \\ \texttt{mismatch} & \text{if } x_i \neq y_j \end{cases} \tag{2.7}$$

Equation 2.6 corresponds to the recursive structure of the *Needleman-Wunsch algorithm* (Needleman and Wunsch, 1970), a widely known algorithm in bioinformatics to compare biological sequences.

Given the example in Figure 5, the value of the current cell is calculated by minimizing over the three possible values, thus

$$d_{2,2} = min \begin{cases} d_{1,1} + w_{align}(R, A) = 2 + 2 = 4 \; (green) \\ d_{1,2} + w_{delete}(R, -) = 1 + 1 = 2 \; (pink) \\ d_{2,1} + w_{insert}(-, A) = 3 + 1 = 4 \; (orange) \end{cases} = 2. \tag{2.8}$$

Given the example calculation in Equation 2.8, the three recursion cases directly correspond to the colored cells in Figure 5. In order to reach the next cell, the cost of the applied rule is added to the value in the previous cell. After calculating all the scores, the minimal value will be chosen to be memoized at $d_{2,2}$ and reused in the next step of the algorithm.

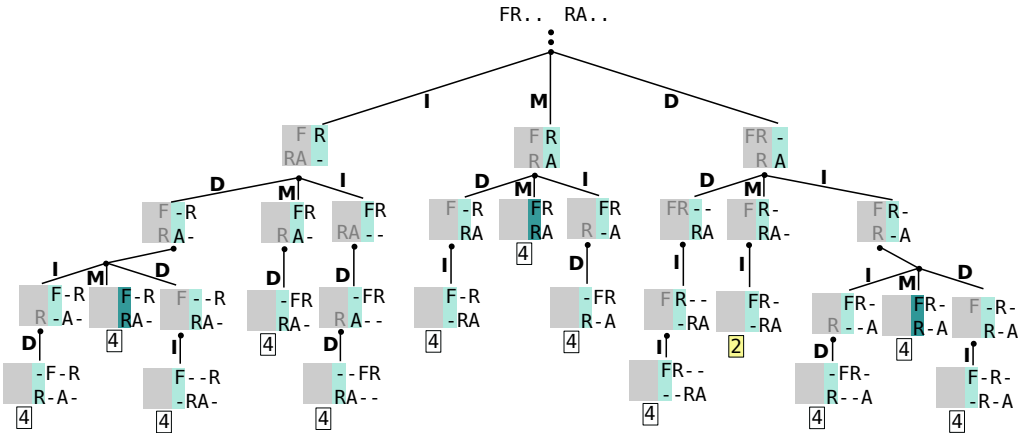


Figure 7: Recursion tree displaying all possible cases when aligning the sequences `fr` (top line) and `ra` (bottom line). Grey areas show subsequences that have been processed, light green areas show currently processed positions and not yet processed characters are shown on its r.h.s. Bold letters at the branches indicate recursion cases of (mis)match (M), insertion (I) and deletion (D). The dark green areas show that the case of aligning the letters F and R appear several times. The white cells indicate distance scores of the branch from the corresponding leaf to the root of the tree. The yellow cell shows the optimal solution.

The score of the overall solution is written in the 'last' cell of the memoization table, $d_{n,m}$ as shown in Figure 6. However, one more step is needed to retrieve the structure of the solution. This is called *backtracking*. Here, one starts from the cell containing the final score tracing back the way of the algorithm. By reconsidering the three possibilities, one follows the way of the optimal score that was used to calculate the value of the current cell as shown in Figure 6. It is also possible to remember for each cell where the optimal score was taken from. This would save time for the backtracking step, however, it needs a second memoization table in order to store the required information.

The tree in Figure 7 depicts possible recursion steps when only aligning prefixes of length two of `friday` and `railway`, namely `fr` and `ra`. As given in the recursion equations, there are three possible cases depicted as the children of a node in the tree: (mis)match M (middle), insertion I (left) or deletion D (right). If less than three children are displayed, the missing cases cannot be applied due to empty input strings. It can be seen that certain cases such as aligning the letters F and R appear several times (dark green area) and thus, their value is stored in the memoization table and then reused when needed.

2.2 Dynamic Programming

The string editing and alignment algorithms as described above are algorithms optimizing towards a certain goal, e.g., minimizing the cost for the total number of edit operations or the distance between the input strings. This kind of optimization algorithms can be solved using DP. Dynamic programming was developed and first described by Bellman

(1952) as a mathematical optimization method. The term 'programming' does not refer to writing code but rather tabulation of intermediate solutions.

The way of tabulating intermediate values leads to a stepwise recursive process. Thus, the same calculations are applied repeatedly to substructures of the input. As a consequence, Bellman (1952) states DP algorithms as sequential decision processes where in each step the algorithm decides for the optimal value based on a given policy. In mathematical optimization, the policy is also called *objective function*. The mechanism is depicted in Figure 5 showing possibilities of how to fill a cell in the matrix which directly corresponds to the recursion step applied to each instance of the problem. The optimal possibility is chosen by the objective function. This is described by *Bellman's principle of optimality* (Bellman et al., 1954) whereas a corresponding formal description is given in Subsection 2.2.2:

An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions.

The principle behind optimization algorithms is to find the optimal solution in a large set of possible solutions. This can be done by first calculating all possible solutions and then filter for the optimal one. Given a DP algorithm, Bellman's principle states that calculating the optimal solution to the problem is equivalent to taking the optimal solution in each step (*optimal substructure*) concatenating subsolutions in order to get the overall solution (*overlapping subproblems*). Thus optimal overlapping subsolutions are composed such that they result in the optimal overall solution, then the final solution does not contain any non-optimal (sub)solutions. Given the recursion equations, subsolutions are hereby calculated several times (e.g. the score for the optimal alignment of just the first two letters of the input strings, R and F as depicted in Figure 5 and Figure 7 on page 21). In order to compute each subsolution at most once, a memoization table is used that stores all subsolutions.

The recursion equations describe the algorithm in a top-down manner, thus starting from the complete sequences and recurring on smaller substrings. However, one can directly calculate the solution in a bottom-up approach, filling the memoization table in an order-preserving way. This means that for all input strings (or structures) the order of elements has to be preserved while filling the table, starting with the first letters of the input strings. Then, it is assured that needed subsolutions for the current step have already been calculated (Cormen et al., 2009). Applying a DP algorithm to more than two inputs is possible, however, the dimension of the memoization table corresponds to the number of input structures, and thus, time and space requirements increase significantly. The creation of a multiple sequence alignment (MSA) to directly compare various sequences is a popular tool in bioinformatics. Due to high resource requirements, MSAs are constructed based on heuristics. More details about the usage of MSAs will be given in Subsection 2.3.1.

2.2.1 Basic Data Structures

Since a key element of dynamic programming is the recursive traversal of the input, this subsection will give an overview and definitions of possible input structures of DP

algorithms. Data structures consist of a finite set of elements S, with $|S|$ being the number of elements in the set. There exist (pairwise) relations $\mathcal{R}$ between the elements of S. Thus, the set of relations between elements of S defines the complete structure. The subsequent definitions list basic data structures and corresponding order relations, also summarized in Table 1.

An often used relation between elements is their order:

Definition 3 (Partial Order, *cf.* Jost (2015))
Given a set S and a binary relation $\mathcal{R}\,(\leq)$ on its elements. The set is a partially ordered set (or poset) if for all elements $a, b, c \in S$ the following holds:

- $a \leq a$ *(reflexivity),*
- *if* $a \leq b$ *and* $b \leq c$ *then also* $a \leq c$ *(transitivity),*
- *if* $a \leq b$ *and* $b \leq a$ *then* $a = b$ *(antisymmetry).*

The definition of the partial order (PO) allows to have pairs of elements $a, b \in S$ to be incomparable. If all pairs $a, b \in S$ are mutually comparable, the order is a *total order (TO)*. The elements of the set S are hereby not further specified. A relation is called a *pre-order* if it is reflexive and transitive. If a relation fulfills reflexivity and transitivity but is additionally symmetric, it is called *equivalence relation (ER)*. A totally ordered finite set $\{s_1, \dots s_n\}$ can also be denoted by a n-tuple $(s_1, \dots s_n)$ or a list $[s_1, \dots s_n]$ on n elements which are by definition totally ordered. Since all of the data structures can be represented as a *graph* in general, its definition is given first:

Definition 4 (Graph)
Let $G = (V, E)$ be a graph with vertex set V and directed edge set $E \subseteq \{\{v, w\} | v, w \in V\}$. Edges directed from the source v to the target node w are denoted as $(v, w) \in E$. Directed graphs are also called digraphs. A graph $G = (V, E)$ is undirected, if its edges are undirected or for each edge $(v, w) \in E$ there exists a corresponding edge $(w, v) \in E$. Then, an edge between nodes $v, w \in V$ is denoted as $\{v, w\} \in E$.

Vertex set and edge set of the graph G can also be denoted by $V(G)$ and $E(G)$ or V_G and E_G, respectively. Edges can be labeled and the label is denoted by $label(e),\ e \in E$. Analogously, nodes in a graph can be labeled, too, denoted by $label(v),\ v \in V$. A graph is called *complete* or a *clique* if $E = V \times V$, also denoted by K_n with $n = |V|$.

Depending on the purpose of the graph data structure, the definition of a directed or undirected version of the graph is essential. If not further specified, graphs defined here are assumed to be simple, thus there is at most one edge between any pair of nodes and no edges from a node to itself (loop).

Definition 5 (Graph Complement)
The complement of a graph $G = (V, E)$ is the graph $\overline{G} = (V, \overline{E})$ such that for all pairs of nodes $u, v \in V$ with $(u, v) \notin E$, there is an edge $(u, v) \in \overline{E}$.

Figure 8 shows an example as the complement of a thin spider graph is a thick spider graph. This is indicated by the node labels as body nodes of the thin spider become feet nodes of the thick spider.

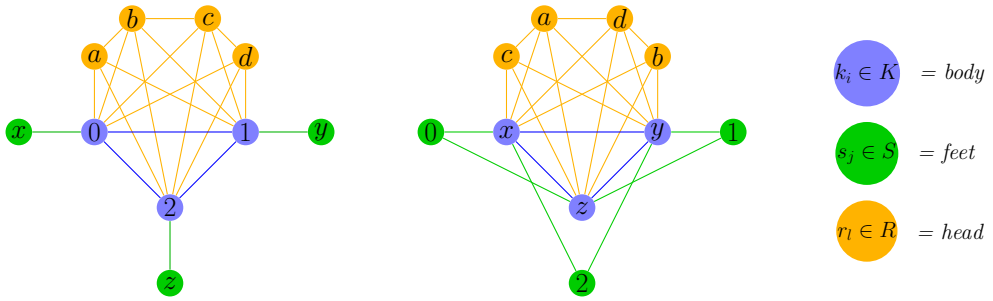


Figure 8: Example for spider graphs (*cf.* Berkemer et al. (2015)) showing a thin spider (left) and a thick spider (right). Spiders are defined based on their node partitions where the body nodes from a clique, the feet an independent set and each of the nodes in the head must be connected to each of the nodes in the body. A thin spider is the complement graph of a thick spider.

Definition 6 (Node Degree)
The degree $deg(v)$ of a node $v \in V(G)$ is the number of incident edges to v, thus $deg(v) = |\{e \in E | v \in e\}|$ for an undirected edge e. If G is a directed graph, one distinguishes between the number of incoming and outgoing edges, called indegree ($indeg(v)$) and outdegree ($outdeg(v)$), respectively.

The degree distribution of a graph shows the number of nodes in G having a certain degree d. Certain types of graphs can be classified based on the shape of their degree distribution such as scale-free networks (Newman, 2010) where the degree distribution follows a power law or spider graphs (Berkemer et al., 2015) where the degree distribution can be used to distinguished different types of nodes, see Figure 8.

Definition 7 (Path & Cycle)
Let $G = (V, E)$ be a directed graph. Then a path P_k in G is a set of k nodes $\{v_1, \dots v_k\}$ and $k-1$ edges such that $(v_i, v_{i+1}) \in E$, $\forall i \in \{1, \dots k-1\}$. The nodes v_1 and v_k are source and sink nodes of the path. A cycle is a path $P_k = (v_1, \dots v_k)$ with an additional edge (v_k, v_1) between the last and the first nodes of the path. Cycles of length k are also denoted by C_k. Paths and cycles in an undirected graph do not have a direction, thus it is possible to walk along the path or cycle in both directions.

Thus, a path in a directed graph is a totally ordered set of nodes whereas the order is defined based on its edge set. A path directly corresponds to the structure of a string or sequence.

Definition 8 (String or Sequence)
A string is a finite sequence of elements of S with S being a finite totally ordered set w.r.t. a binary relation R. The Kleene star of S is the set S^* of all strings composed out of the concatenation of elements taken from S, cf. Droste and Kuich (2009).

The graph representation of a string is a simple directed path $P_{|S|}$ on $|S|$ nodes.

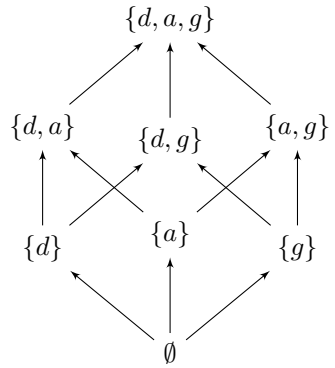


Figure 9: Hasse diagram for set inclusion ($\subset$) whose structure is a DAG. The source node is the empty set $\emptyset$ and the sink depicts the complete set $\{d, a, g\}$. An edge between two nodes indicates its source node's label being a proper subset of the sink node's label. Transitive edges are not drawn within the Hasse diagram but the subset relation still holds for nodes connected by a path. If two nodes are not connected, they are incomparable.

Definition 9 (Subgraph and Induced Subgraph)
Given a graph $G = (V, E)$ a subgraph H of G is a graph on a subset of vertex and edge set, thus $H = (V_H, E_H)$ with $V_H \subseteq V$ and $E_H \subseteq E$. An induced subgraph additionally requires that if $\{v, w\} \in E$ and $v, w \in V_H$ then $\{v, w\} \in E_H$, $\forall v, w \in V_H$. An induced subgraph is also denoted by $H = (V, E)[V_H]$.

Induced subgraphs show the structure of a graph in a certain neighborhood of nodes. For some types of graphs, certain kinds of induced subgraphs are not allowed, e.g. P_4s in cographs (Hellmuth and Wieseke, 2015). Induced P_4s also appear in spider graphs, e.g., the path $0 - x - y - 1$ in the thick spider in Figure 8.

Definition 10 (Connected Component)
A connected component of a graph $G = (V, E)$ is an induced subgraph $H = (V_H, E_H)$ of G such that there is a path between every pair of nodes in V_H.

If a graph consists of one connected component , it is called *connected*, otherwise it is disconnected.

Definition 11 (Directed Acyclic Graph)
A directed acyclic graph (DAG) is a directed graph that does not contain any cycles.

A DAG $G = (V, E)$ represents a partial order where pairs of nodes $v, w \in V$ are incomparable if there is no path between them. Partial orders are frequently displayed by *Hasse diagrams* (Hasse, 1985) whose structure is a DAG as shown in Figure 9.

Definition 12 (Ordered Trees and Ordered Forests)
An ordered forest $F = (V, E)$ is a DAG where the set of nodes can be partitioned into three disjoint sets such that

structure	relations	example for $\mathcal{R}$
string/sequence	TO	$\leq$
tree/forest	PO	sibling $\lhd$ and parent $\preceq$ order
set	ER	collection, complete graph
DAG	PO	directed edges
graph	PO	(un)directed edges

Table 1: Table listing basic data structures and corresponding underlying order relations.

- $\mathcal{W} = (r_1, \dots r_r)$ *is the sequence of roots of the trees in the forest. Roots have only outgoing edges and are thus source nodes,*
- *the set of inner nodes* $\mathcal{I}$ *having exactly one ingoing edge and at least one outgoing edge,*
- *the set of leaves* $\mathcal{L}$ *with elements having exactly one incoming and no outgoing edge and thus, are sinks.*

An ordered forest F can be composed out of several connected components, which defines an ordered set of trees $\{T_1, \dots T_r\}$ in the forest. A rooted ordered tree with vertex set V is uniquely defined by two mutually exclusive partial order relations: the *ancestor order* $\prec$ defined such that $x \preceq y$ whenever y is located on the path from the root to x, and the *sibling order* $\lhd$ defined in terms of the ordering of the children of each vertex: The last common ancestor of two nodes u and v in a tree is a node l being in the set of nodes in the intersection of paths from the root to the nodes u and v where l is the node in the set being closest to u and v. Thus, for two vertices x and y that are incomparable w.r.t. $\prec$, let w be their last common ancestor and u and v be the distinct children of w such that $x \preceq u$ and $y \prec v$. Then $x \lhd y$ if and only if $u \lhd v$ (Gärtner and Stadler, 2019). By construction, two vertices are either identical or comparable w.r.t. either the ancestor or the sibling order. The observation extends to ordered forests, where the sibling order is extended such that vertices from any two constituent subtrees are are always comparable w.r.t. the sibling order.

Furthermore, there are two distinct types of orders defined on the nodes of a tree or forest: The *preorder* starts at the (leftmost) root of the tree or forest going always first to its left child or, if the node is a leaf, further to its right sibling. In this way, the nodes of the tree are mapped into a sequence. The *postorder* starts at the leftmost child of a tree or forest and continues to its right sibling. Thus, children of a node get listed before their parent node. Edges in a tree are directed from the root towards the children. If the direction is clear due to the placement of the root, edges will be displayed as undirected edges due to simplicity.

Table 1 summarizes the most basic data structures described in this chapter. All the data structures can be seen as sets of elements with a pairwise relation $\mathcal{R}$ which defines the structure. A set is a collection of elements without further structural properties. Therefore, one can depict a set as a complete graph such that all pairs of elements are in a pairwise relation. As this relation is reflexive, transitive and symmetric, it is an ER.

2.2.2 Mathematical Formulation

Dynamic programming algorithms can be easily formulated using recursion equations. However, this description does not include any requirements on the input structure or the equations themselves in order to describe a dynamic programming (DP) algorithm. A more complete definition is formulated by Tendeau (1998) and will be given below after a few basic definitions.

As mentioned above, the principle of a DP algorithm is the composition of optimal subsolutions. Thus, in order to mathematically describe DP algorithms, we need two operators to (i) compose (concatenate) subsolutions and (ii) choose the optimal one (objective function). Such a combination of operators is given by the mathematical structure of a *semiring* which is composed out of two *monoids*.

Definition 13 (Monoid, *cf.* Jost (2015))
A monoid M is a structure $(M, \cdot, e)$ such that

- *the binary operation $\cdot : M \times M \to M$ maps a pair of elements $k, l \in M$ to their product $k \cdot l \in M$,*
- *$\cdot$ is associative, thus $k \cdot (l \cdot m) = (k \cdot l) \cdot m$, for all $k, l, m \in M$,*
- *e is the neutral element such that $e \cdot m = m \cdot e = m$, for all $m \in M$.*

The monoid is additionally called commutative if $k \cdot l = l \cdot k$, for all $k, l \in M$.

The nonnegative natural numbers $\mathbb{N}_{\geq 0}$ with addition $+$ and 0 as neutral element are the monoid $(\mathbb{N}_{\geq 0}, +, 0)$. Taking two monoids together will result in a semiring:

Definition 14 (Semiring, *cf.* Kuich (1997))
A semiring R is a structure $(R, \oplus, \otimes, 0, 1)$ such that

- *$(R, \oplus, 0)$ is a commutative monoid,*
- *$(R, \otimes, 1)$ is a monoid,*
- *multiplication is distributive over addition from both sides, i.e.*
  $k \otimes (l \oplus m) = (k \otimes l) \oplus (k \otimes m)$ and
  $(k \oplus l) \otimes m = (k \otimes m) \oplus (l \otimes m)$, for all $k, l, m \in R$,
- *0 is absorbent for the multiplication, i.e., $0 \otimes m = m \otimes 0 = 0$, for all $m \in R$.*

The natural numbers $\mathbb{N}$ with addition $+$ and neutral element 0 and multiplication with neutral element 1 are the semiring $(\mathbb{N}, +, \cdot, 0, 1)$. A very basic semiring is the boolean semiring $(\{\text{TRUE}, \text{FALSE}\}, \vee, \wedge, \text{FALSE}, \text{TRUE})$. Widely used semirings in DP algorithms are the tropical $(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$ and the arctic $(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$ semiring. However, calculation of probabilities or just enumerating the search space require distinct semirings such as the Viterbi semiring $(\mathbb{R}_0^1, \max, \cdot, 0, 1)$ defined on the real numbers in the interval $[0, 1]$ or the counting semiring $(\mathbb{N}, +, \cdot, 0, 1)$. More detailed descriptions of semirings can be found in Goodman (1998) and Kuich (1997). Semirings mentioned above are also listed in Table 2.

Tendeau (1998) gives a definition for DP algorithms:

Name	Semiring
boolean	$(\{\text{TRUE}, \text{FALSE}\}, \vee, \wedge, \text{FALSE}, \text{TRUE})$
counting	$(\mathbb{N}, +, \cdot, 0, 1)$
tropical	$(\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$
arctic	$(\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)$
Viterbi	$(\mathbb{R}_0^1, \max, \cdot, 0, 1)$
probabilities	$(\mathbb{R}, +, \cdot, 0, 1)$

Table 2: Summary of well-known and widely used semirings used for DP algorithms.

Definition 15 (Dynamic Programming, Tendeau (1998))
A dynamic programming problem over input $d \in D$ can be described by the following elements:

- *a recursively defined function: $F : D \to R$ with R being a semiring $(R, \oplus, \otimes, 0, 1)$,*
- *a partial pre-order $\prec$ on D,*
- *a function $\phi : D \times D^* \to R$ being the contribution in each recursive step where $\phi(d, d_1, \dots d_k)$ is the weight of the current step for d depending on the previous values d_1 to d_k,*
- *a binary relation $\diamond \subset D \times D^*$ that associates with each element $d \in D$ some predecessor sequences in $(d_1, \dots, d_k) \in D^*$ that contain only elements smaller than d w.r.t. $\prec$, i.e., $\forall d \in D, \forall (d_1, \dots, d_k) \in D^*, d \diamond (d_1, \dots, d_k) \Rightarrow d_i \prec d$, for all $i \in \{1, \dots k\}$.*

Given the above definition, a DP algorithm can be written as (Tendeau, 1998):

$$\forall d \in D,\ F(d) = \bigoplus_{d \diamond (d_1, .., d_k)} \left(\phi(d, d_1, .., d_k) \otimes \bigotimes_{i \in \{1, .., k\}} F(d_i) \right) \tag{2.9}$$

The above equation summarizes the structure of DP algorithms. The recursive equation F includes a case distinction over possible structures of the current input. The chosen case then defines the predecessor sequence to recurse on as well as the score contribution of the current recursive step. The calculation of scores is applied on the semiring operations addition $\oplus$ and multiplication $\otimes$. This is reflected in the first part of Equation 2.9 $\bigoplus_{d \diamond (d_1, .., d_k)\}} (\dots)$ where possible cases of the recursive function are summarized by addition. This is also called objective function and corresponds to, e.g., max or min in the application of string alignment. The second part of Equation 2.9 is composed out of the recursive step $\bigotimes_{i \in \{1, .., k\}} F(d_i)$ that applies F to smaller instances of the problem, thus, predecessor sequences. In the case of string alignment, each case only leads to one predecessor sequence, as shown below. Depending on the case, the scoring function $\phi(d, d_1, .., d_k)$ results in a score on the current instance of the problem The score and result of the recursive step are summarized by multiplication $\otimes$ which corresponds to addition $+$ in the Needleman-Wunsch algorithm.

Consider the recursion tree in Figure 7 on page 21 as an example. The current input is the pair of word prefixes `fr` and `ra`. The recursion equation of the Needleman-Wunsch algorithm consist of the cases align, insert, delete and empty where align is divided into match and mismatch within the scoring function and the empty case can only be applied to an empty input. The scoring is based on the semiring $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$. The first step of the program is to minimize over all rules that can be applied to the current input:

$$\begin{aligned} F(fr, ra) &= \min\{F_{align}(fr, ra), F_{delete}(fr, ra), F_{insert}(fr, ra)\} \\ &= \min\{w_{align}(r, a) + F(f, r), \\ &\quad w_{delete}(-, a) + F(fr, r), \\ &\quad w_{insert}(r, -) + F(f, ra)\} \end{aligned}$$

The recursive calls behave analogously, e.g., for the case $F(f, r)$:

$$F(f, r) = \min\{w_{align}(f, r) + F(\epsilon, \epsilon), w_{delete}(-, r) + F(f, \epsilon), w_{insert}(f, -) + F(\epsilon, r)\}$$

The case $F(\epsilon, \epsilon)$ will now recurse on the empty rule and halt. The cases of the recursion equation above directly correspond to the branches of the recursion tree in Figure 7 on page 21. As the string alignment algorithm only recurses on one smaller instance of the input, there are no further summands to consider. However, there exist algorithms recursing on more than one instance, which will be described in Subsection 2.2.4.

2.2.3 From Recursion Equations to Formal Grammars

Recursion equations as shown for the Needleman-Wunsch algorithm in Equation 2.6 on page 19 are a well-known tool to describe DP algorithms. They explicitly include the memoization table such that indices of the matrix match with positions in the input structures, as explained by Equation 2.8 and Figure 6 on page 20.

Another way to describe DP algorithms is to use *formal grammars* which was adapted in many bioinformatics applications (Dong and Searls, 1994; Searls, 1992; Durbin et al., 1998) but is also widely used in computer science and linguistics to describe languages and classes thereof (Chomsky, 1959; Goodman, 1998; Hopcroft and Ullman, 1979). The definition of dynamic programming given by Tendeau (1998) is described using recursion equations as well as formal grammars.

The most difficult issue in this context is to identify a sufficiently generic representation of the search spaces on which the individual application problems are defined. Following Giegerich (2000), a natural starting point is a rewriting system which defines instances of the input that can possibly be replaced by intermediate structures. This leads to the definition of a formal grammar $\mathcal{G}$ which specifies a set of rules on how to generate or read strings of the corresponding *language* $L(\mathcal{G})$. The benefit of using formal grammars compared to explicit recurrences is mostly defined in the high-level description of the solution space by the problem at hand. Formal grammars work on an input alphabet, the set Σ. Words of the language $L(\mathcal{G})$ are elements of Σ^*. The *empty word* is denoted by $\epsilon \notin \Sigma$. Regarding alignments, an additional symbol $- \notin \Sigma$ denotes the empty word, too. However, there is a clear distinction between the symbols due to practical reasons:

given the string alignment in Figure 6 on page 20, gap symbols are a key element of its representation such that alignment columns can clearly be distinguished and indels are made explicit. Removing the gap symbols results in the original input word using the function π defined in Definition 2 on page 17. Thus, the '$-$' symbol is used as an explicitly written empty word and the symbol 'ϵ' is used to denote the empty input.

Definition 16 (Weighted Formal Grammar, Tendeau (1998))
A weighted context-free formal grammar (CFG) on a semiring $(R, \oplus, \otimes, 0, 1)$ *is a 5-tuple* $\mathcal{G} = (N, \Sigma, P, S, w)$ *with* $-, \epsilon \notin \Sigma$ *denoting the empty word where*

- N *is a finite set of non-terminal symbols,*
- Σ *is a finite set of terminal symbols,*
- $P \subseteq N \times (N \cup T)^*$ *is a finite set of production rules for* $T = \Sigma \cup \{-, \epsilon\}$*,*
- $S \in N$ *is the start symbol,*
- $w \colon P \to R$ *is a weight function.*

In the following, non-terminals are denoted by capital letters and terminals by lower-case letters. Greek letters denote words of terminals and non-terminals. The production rules introduced above are of the form $A \to \alpha$. If the weight of a production rule is $w(A \to \alpha) = r$, with $r \in R$, we shortly write $A \xrightarrow{r} \alpha$.

Production rules of a grammar $\mathcal{G}$ can fail. If a rule fails, it does not produce a result on the right-hand side. In an implementation, right-hand sides typically yield sets or lists of results, where a failure of a rule naturally leads to an empty set or an empty list. In the algebraic dynamic programming (ADP) framework described in Subsection 2.2.7, rules are either executed as a whole or fail completely. A rule failing partially is not possible.

Definition 17 (Regular Weighted Formal Grammar)
Let $\mathcal{G} = (N, \Sigma, P, S, w)$ *be a weighted formal grammar. If the productions are of the type* $P \subseteq \big(N \times TN \cup N \times T\big)$*, then* $\mathcal{G}$ *is called right-linear. Left-linear grammars are defined analogously with productions of the form* $P \subseteq (N \times NT \cup N \times T)$*. Both, left- and right-linear grammars are called regular.*

Here, weighted regular grammars are a subset of weighted context-free grammars as given in the Chomsky hierarchy (Chomsky, 1959) defining classes of languages which specify certain properties of the language and corresponding grammars. DP algorithms can be applied not only to just one input structure but also to two or more independent inputs. As described in Tendeau (1998), a DP algorithm can be described by recursion equations, formal grammars and finite automata. In the context of finite automata, DP algorithms applied to one input structure are denoted as *single-tape* and two or more inputs as *multi-tape* algorithms (see also Subsection 2.2.4). For all algorithms considered here, the input tape is immutable.

Definition 18 (Multi-tape Inputs)
A multi-tape input is a n-vector of input tapes. In a multi-tape grammar $\mathcal{G}_n$ *operating on n words, production rules can be specified to n tapes and thus the i'th element of the*

vector is operated on by symbols in the i'th position of each rule of the grammar where rules are analogously specified on vectors.

A formal grammar will be in general denoted by $\mathcal{G}$ and the number of tapes will only be specified when needed. The definitions for the set of non-terminals N, terminals T, the alphabet Σ and start symbol S are defined for a single tape, thus the set of non-terminals for all n tapes together is a subset of all possible combinations, N^n. The production rules are defined by $P^n \subseteq N^n \times (N^n \cup T^n)$ which forbids to have a non-terminal on one tape be combined with a terminal on another tape.

In the example for pairwise alignments below, inputs are 2-dimensional vectors. A simple version of the Needleman-Wunsch algorithm written as a weighted formal grammar $\mathcal{G}_2 = (\{X\}, \Sigma, P, X, w)$ with the set of production rules is given by:

$$\binom{X}{X} \to \underbrace{\binom{X}{X}\binom{a}{a}}_{align} \mid \underbrace{\binom{X}{X}\binom{a}{-}}_{delete} \mid \underbrace{\binom{X}{X}\binom{-}{a}}_{insert} \mid \underbrace{\epsilon}_{empty} \tag{2.10}$$

The grammar is defined on two input sequences and thus has a two-line symbol on the left-hand side, the two-tape non-terminal $\binom{X}{X}$. Here, X is a non-terminal of the grammar, a a terminal character, ϵ the empty string and '$-$' the "no input read"-symbol. The left-linear grammar is defined on the cases Xa or $X-$. Two-tape terminals, e.g. $\binom{-}{a}$, act on individual inputs, here not moving on the first input, but reading a character "into" a on the second input. The operations of the grammar can be explained for each symbol individually, while grouping into columns indicates that the symbol on each line, and thereby input, operates in lockstep with the other symbols in the column. Multiple tapes are not processed independently, i.e., 'actions' are taken concurrently on all tapes. Since we consider the insertion of a gap symbol also as an 'action', it is not allowed to insert gap symbols on all the tapes at the same time. Hence, in each step at least one of the tapes is 'consuming' an input character. The algorithm is therefore guaranteed to terminate. There is no explicit start symbol, however, one could add a rule $\binom{S}{S} \to \binom{X}{X}$. The weight function w assigns a weight to each production rule as it was shown in the example in Figure 6 on page 20 aligning the words `friday` and `railway`.

The language of a formal grammar $L(\mathcal{G})$ is the set of all possible words that can be generated by applying a finite sequence of production rules of the grammar. Any formal weighted grammar $\mathcal{G} = (N, \Sigma, P, S, w)$ over a semiring $(R, \oplus, \otimes, 0, 1)$ and with $T = \Sigma$ induces a *derivation relation* $\Rightarrow_{\mathcal{G}} \subseteq (N \cup T)^+ \times (N \cup T)^*$ as follows:

$$\alpha X \beta \Rightarrow_{\mathcal{G}} \alpha\phi\beta \text{ iff } (X \to \phi) \in P, \text{ for } X \in N, \;\; \alpha, \beta, \phi \in (N \cup T)^*$$

If $\alpha X \beta \Rightarrow_{\mathcal{G}} \alpha\phi\beta$ with the rule $(X \to \phi) \in P$ and $w(X \to \phi) = r$ then we also write $\alpha X \beta \overset{r}{\Rightarrow}_{\mathcal{G}} \alpha\phi\beta$. Applying a grammar $\mathcal{G}$ on an input v is called a *derivation* of the grammar:

Definition 19 (Derivation of $\mathcal{G}$, *cf.* Droste and Kuich (2009))
Let $\mathcal{G} = (N, \Sigma, P, S, w)$ be a weighted grammar over a semiring $(R, \oplus, \otimes, 0, 1)$. A derivation δ of $\mathcal{G}$ on an input v is a sequential application of the derivation relation starting with start symbol S and ending with v:

$$\delta\colon S \overset{r_0}{\Rightarrow} \alpha_0 \overset{r_1}{\Rightarrow} \ldots \overset{r_m}{\Rightarrow} v; \qquad \text{for } r_i \in R, \;\; \alpha_i \in (N \cup \Sigma)^*, \;\; v \in \Sigma^*$$

The derivation δ of a grammar $\mathcal{G}$ on an input v produces a sequence of rule applications until v is reached.

Given a weighted formal grammar $\mathcal{G}$ and an input v, there may exist various derivations ending with v. Thus one can combine the weights of single derivations into a *weighted language or series*:

Definition 20 (Series of $\mathcal{G}$)
Let $\mathcal{G} = (N, \Sigma, P, S, w)$ be a grammar over a semiring $(R, \oplus, \otimes, 0, 1)$ and $\delta\colon S \overset{r_0}{\Rightarrow} \alpha_0 \overset{r_1}{\Rightarrow} \ldots \overset{r_m}{\Longrightarrow} v$ a derivation of $\mathcal{G}$ on input v. The weight of derivation δ is

$$w(\delta) = \bigotimes_{r=0}^{m} r_i \,.$$

The series generated by such a grammar is the function $L(\mathcal{G})\colon \Sigma^* \to R$ defined by

$$L(\mathcal{G})(v) = \bigoplus (w(\delta) \mid \delta \text{ derivation of } \mathcal{G} \text{ on } v) \,.$$

The series is a function describing the language of a weighted grammar. In the unweighted case, the semiring R would be the boolean semiring and thus, return true if a word is in the language and false otherwise.

For the development and application of formal grammars describing dynamic programming problems on strings, several approaches have been developed in recent years (Giegerich, 2000; Höner zu Siederdissen et al., 2015a; Höner zu Siederdissen, 2012; Sauthoff et al., 2011; Sauthoff et al., 2013). The formulation of DP algorithms using formal grammars results in a separation of the rule description, the weight function and the memoization. This approach is called ADP (Giegerich, 2000; Giegerich and Meyer, 2002) and will be defined in Subsection 2.2.7.

2.2.4 From Formal Grammars to Parsers

Another well-known problem that can be solved using dynamic programming is the matrix-chain-multiplication (Cormen et al., 2009). Given two matrices M, N with dimensions $a \times b$ and $b \times c$, respectively. Then, the number of single operations needed to calculate the product $M \times N$ of both matrices is $a \cdot b \cdot c$. Now assume a sequence of matrices $M_1 \ldots M_n$ to be multiplied. The goal is to find a bracketing of the matrices such that the number of single operations needed is minimized when calculating the matrix product.

Let $i, k, j \in \{1, \ldots n\}$ be indices of the matrices and $m(i, j)$ be the minimal number of operations needed to calculate a product including matrices i to j with $i < j,\ i, j \in \{1, \ldots n\}$. Let $r(i)$ denote the number of rows of matrix i and $c(i)$ the number of its columns. Hence, the number of multiplications for the matrix product $M_i \cdot M_{i+1}$ is $r(i) \cdot r(i+1) \cdot c(i+1)$ as $c(i) = r(i+1)$ is the number of columns of M_i and the number of rows of M_{i+1}. Then the recursive solution is given by:

$$m(i, j) = \min_{i \le k < j} \big(m(i, k) + m(k+1, j) + r(i) \cdot r(k+1) \cdot c(j)\big) \tag{2.11}$$

with initialization defined as $m(i, i) = 0$. The corresponding formal grammar is a *context-free grammar* (CFG) as there are two parallel distinct recursive calls given in the recursion

equation. The corresponding case in Equation 2.12 is called 'mult' with two non-terminal symbols MM. The terminal m describes the case when a matrix is taken as single element in the current step and ϵ denotes the empty input. Note that the 'mult' case combines two subsolutions opposite to the usual choice of the currently optimal subsolution done in each step of the DP algorithm. Let $\mathcal{G} = (\{M\}, \Sigma, P, M, w)$ be the grammar for the matrix-chain-multiplication problem with the following production rules:

$$M \to \underbrace{m}_{single} \mid \underbrace{MM}_{mult} \mid \underbrace{\epsilon}_{done} \tag{2.12}$$

The Needleman-Wunsch algorithm described by the grammar in Equation 2.10 and the matrix-chain-multiplication given in Equation 2.12 not only differ concerning their classes of formal grammars but also in the calculation of scores. The string alignment algorithm has a scoring system that scores grammar rules differently in order to distinguish distinct (evolutionary) events. Only the distinction between match and mismatch cases is based on the sequence data itself. In contrast, the scoring of the matrix-chain-multiplication problem is completely based on the dimensions of the matrices such that distinct grammar rules are not required to have different scores, i.e., all the rule based scores are set to 1. A further difference between the Needleman-Wunsch algorithm and the matrix-chain-multiplication is the number of input structures. While the latter is applied to only one sequence of matrices, the former compares two strings, thus is applied to two input structures.

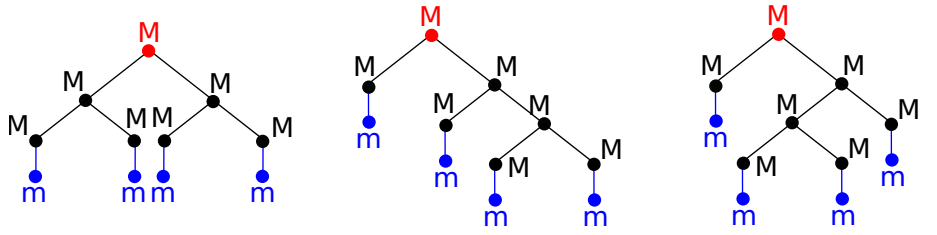
Chomsky and Schützenberger (1963) added weights to context-free languages and obtained algebraic power series. This formalism generalizes context-free grammars in such a way that the productions in a context-free grammar (CFG) correspond to a set of equations in an algebraic system. The weighted context-free language is defined as the least solution of the equation system. Petre and Salomaa (2009) give a detailed survey on the topic.

Given the description of DP algorithms as formal grammars, one can use properties of formal language theory to extend notions and applications of DP algorithms. Formal grammars are closely related to finite automata and complement the concepts of generating and parsing words of a language (Hopcroft and Ullman, 1979). A *parser* therefore is considered to check if the input is part of the language. Parsers can also be combined with weight structures such as semirings (Goodman, 1998). Describing a DP algorithm using weighted formal grammars can be interpreted as parsing the input in an optimal way.

The parser generates a *parse tree* in order to depict the way of the program. Parse trees and derivations of a grammar are two distinct ways of describing the sequence of grammar rules applied to an input. Parse trees are tree structures with specific node labels:

Definition 21 (Parse Tree, *cf.* Hopcroft and Ullman (1979))
A parse tree is a tree depicting a derivation of a grammar $\mathcal{G} = (N, \Sigma, P, S, w)$ on an input v s.t. the following holds:

- *every vertex has a label such that*
  - *the root has the label S,*



$(M_a \cdot M_b) \cdot (M_c \cdot M_d)$ $M_a \cdot (M_b \cdot (M_c \cdot M_d))$ $M_a \cdot ((M_b \cdot M_c) \cdot M_d)$

Figure 10: Example parse trees (top) for the grammar $\mathcal{G}$ in Equation 2.12 and the input $M_{abcd} = M_a M_b M_c M_d$ as product of four matrices. Red nodes stand for the start symbol, black nodes and edges indicate the 'mult' rule and blue nodes and edges show the 'single' rule of the grammar. ϵ-rules are omitted. The bottom line indicates the bracketings for the matrix products resulting from the above parse trees, respectively.

  - *inner nodes have labels in N,*
  - *leaves have labels in T,*

- *if an inner vertex has a label $X \in N$ and its children have labels $C_1, \dots C_k \in N \cup T$ then there exists a production rule in P with $X \to C_1, \dots C_k$,*

- *if a leaf has label ϵ it is the only child of its parent node.*

As an example, consider a product of four matrices, $M_{abcd} = M_a M_b M_c M_d$ as an input to the grammar $\mathcal{G}$ in Equation 2.12. Then, a possible derivation of $\mathcal{G}$ on P is:

$$\delta\colon M \overset{w_{mult}}{\Longrightarrow} MM \overset{w_{single}}{\Longrightarrow} mM \overset{w_{mult}}{\Longrightarrow} mMM \overset{w_{single}}{\Longrightarrow} mmM \overset{w_{mult}}{\Longrightarrow} mmMM \overset{w_{single}}{\Longrightarrow} mmmM \overset{w_{single}}{\Longrightarrow} mmmm \overset{w_{done}}{\Longrightarrow} \epsilon$$

This derivation corresponds to the parse tree in the middle of Figure 10.

Given matrix dimensions for the matrices in $M_{abcd} = M_a M_b M_c M_d$, one can calculate a score for each parse tree which equals the number of single operations needed to calculate M_{abcd}. A possible way to solve the matrix-chain-multiplication problem would be to calculate all possible parse trees (or derivations) for an input and choose the one corresponding to the minimal score. This corresponds to calculating the series in Definition 20 using the tropical semiring $(\mathbb{R}, min, +, \infty, 0)$.

In comparison, DP algorithms do not calculate all possible parse trees but rather all pairs of possible subsolutions and recurse on the best one as defined in Equation 2.9. Thus, the only interest lies in the parse tree corresponding to the optimal solution in order to extract the derivation and thus, the structure of the solution (e.g. by backtracking the memoization table).

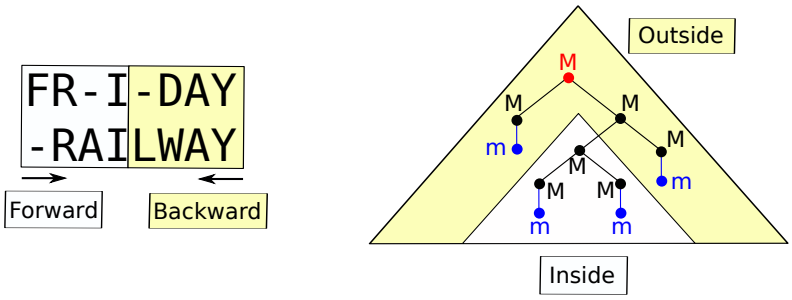


Figure 11: Example of the forward-backward (l.h.s) and inside-outside calculations (r.h.s). Forward and inside values are calculated based on a well-defined substructure (white) whereas backward and outside probabilities are the based on the complementary structure (yellow).

2.2.5 Calculating Probabilities

Instead of calculating the solution with the optimal score, it is also possible to calculate the most probable path through the memoization table. This can be done by applying a grammar $\mathcal{G} = (N, \Sigma, P, S, w)$ where the weight function $w : P \to R$ is defined by the Viterbi semiring $(\mathbb{R}_0^1, \max, \cdot, 0, 1)$ based on real numbers in the interval $[0, 1]$. and the weight function defines probabilities for each production rule. The optimal score is based on the maximal probability. This is called the *Viterbi algorithm* (Durbin et al., 1998; Viterbi, 1967).

In order to calculate the total probability for an input v to be derived by a grammar $\mathcal{G}$, one can use the inside semiring $(\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$. Here, rule probabilities are multiplied within one derivation and scores of all derivations are summed up to get the final result (Durbin et al., 1998; Goodman, 1998; Goodman, 1999). Since it is possible to use semirings as weight structures for distinct kinds of grammars, different algorithms can be defined based on the grammar-semiring combination. If the inside semiring is applied as a weight structure to a regular grammar, the algorithm is called *forward algorithm*. If the grammar is a CFG, the algorithm name matches the semiring, thus it is called *inside algorithm* (Durbin et al., 1998; Goodman, 1998).

While the inside or forward probabilities define the probability of a certain (sub)derivation to be in place, the backward or outside probabilities define the probability of derivations occurring with exactly the inside part missing as shown in Figure 11. The corresponding algorithms are called *backward algorithm* for regular grammars and analogously, *outside algorithm* for CFGs. Forward and inside probabilities are calculated with a DP algorithm starting from the smallest substructure ending when the structure is traversed completely. This is based on the order of elements in the input structure. In order to calculate backward or outside probabilities, one needs to reverse the order relation of the input. Thus, the backward algorithm starts with a complete input and stops when the input is traversed completely, reaching the smallest (first) element. The outside algorithm is additionally more complicated as it requires two bounds between the inside and the outside part as depicted in Figure 11. Analogously to the forward or inside algorithm, being applied to all possible subsolutions, the backward or outside

algorithm is also applied to all possible substructures and the order of calculation is reversed. Combining the algorithms, thus applying forward-backward or inside-outside on an input v, one can deduce probabilities for each element in v to be in a certain position of the resulting structure. The description of inside-outside algorithm in the framework of ADP is described in Subsection 2.2.7 including more detailed descriptions of the algorithm. Chapter 3 shows results of the inside-outside algorithm applied to trees and forests as input structures. More detailed descriptions of the forward-backward and inside-outside algorithms can be found in Durbin et al. (1998) for bioinformatics applications and in Goodman (1998) and Goodman (1999) regarding parsers in the field of linguistics.

2.2.6 Index Structures

The order of elements of the input structure is a key element for the application of DP algorithms. Given strings as an input, their totally ordered set of elements defines the order of program execution. However, other input structures, such as trees or forests, require to first determine the order of dependencies in the input before running the program. This is called the *index structure* of the input. For trees and forests, popular index structures are preorder and postorder on trees, as described in Subsection 2.2.1 and in Chapter 3. The index structure is implicitly combined with the input and the recursion equations of the program. Irrespective of time and space requirements, one could also first create a dependency graph and use connected components and topological sorting to determine the order of the program for more complex inputs. There exist more time and space efficient ways of calculating the optimal order of items (Goodman, 1998) which also depends on the kind of algorithm applied to the input. For sets, a possible structure is the power set and its partial order defined by subset inclusion. Set structures together with DP algorithms have been discussed in Höner zu Siederdissen et al. (2015b). Chapter 4 gives a more detailed description about order relations on input structures, especially for alignments, and gives characterizations of possible input structures.

2.2.7 Algebraic Dynamic Programming

Algebraic dynamic programming (ADP) (Giegerich, 2000; Giegerich and Meyer, 2002) is designed around the idea of higher order programming and starts from the realization that a dynamic programming algorithm can be separated into four parts: (i) grammar, (ii) algebra, (iii) signature, and (iv) memoization. These four components are devised and written separately and combined to form possible solutions. As mentioned in Subsection 2.2.6, each input requires an additional index structure to determine the order of program execution on its elements. Even though implementing this step is usually included in the process of reading the input, one needs to specify the index structures required for the DP algorithm.

One advantage of ADP is that the decoupling of the individual components makes each individual component much easier to design and implement. Another advantage is that different components can be reused easily. This is in particular true for the grammar, which defines the search space. Hence, it needs to be specified only once. The grammar specifies how to decompose the input based on the production rules. The *scoring algebra* specifies the cost function and the selection criterion, including the objective function,

count	*eval*
$\text{align}(s, \binom{u}{v}) = s$	$\text{align}(s, \binom{u}{v}) = s + w_{align}(u, v)$
$\text{delete}(s, \binom{u}{-}) = s$	$\text{delete}(s, \binom{u}{-}) = s + w_{delete}(u, -)$
$\text{insert}(s, \binom{-}{v}) = s$	$\text{insert}(s, \binom{-}{v}) = s + w_{insert}(-, v)$
$\text{empty}(\epsilon) = 1$	$\text{empty}(\epsilon) = 0$
$\text{choice}([x_1, \ldots x_n]) = x_1 + \ldots + x_n$	$\text{choice}([x_1, \ldots x_n]) = h(x_1, \ldots x_n)$
with $u, v \in \Sigma$ and $s \in R$	with $u, v \in \Sigma$ and $s \in R$

Figure 12: Two different scoring algebras written for the grammar of string alignment. Here, $s \in R$ is the score, Σ the input alphabet, $-$ the gap symbol and ϵ the empty string. The score contributions $w(\,.\,,\,.\,)$ evaluate the edit operations, here (mis)matches, deletions and insertions, depending on the values of terminal symbols. The choice function formalizes the selection criterion, or objective function applied in each recursion step. In the *count* algebra (left) it returns the sum of the alternatives. In the *eval* algebra (right), the choice function h is usually defined as max or min to retrieve the optimal value.

and thus defines formally what is considered as an "optimal" solution. Based on the objective function, the optimal solution is selected from the possible solutions. The algebra is based on a semiring and the scope of the evaluation algebra is quite broad, see also Subsection 2.2.2. It can be used e.g. to count and list prioritized (sub)optimal solutions. The *signature* is used as a connecting element between grammar and algebra such that they can be reused and exchanged independently from each other as long as they fit to the signature. The subsolution to each instance of the problem is stored in the memoization table such that it can be reused when needed. In this section, the four parts of ADP will be shortly explained using string alignment as an example whose grammar has already been given in Equation 2.10 on page 31.

Pairwise string alignment is a two-tape algorithm, thus the grammar is given by $\mathcal{G}_2 = (N, \Sigma, P, S, w)$ with $N = \{X\}$, $S = X$ and the production rules given below. Alignment of biological sequences is defined on the alphabet $\Sigma = \{A, C, G, T\}$ when aligning DNA sequences (for more details on DNA, see Chapter 5).

$$\binom{X}{X} \rightarrow \underbrace{\binom{X}{X}\binom{a}{a}}_{align} \mid \underbrace{\binom{X}{X}\binom{a}{-}}_{delete} \mid \underbrace{\binom{X}{X}\binom{-}{a}}_{insert} \mid \underbrace{\epsilon}_{empty} \tag{2.10}$$

The grammar defines 4 different cases: (i) matching the current positions of the input strings, (ii)-(iii) inserting a gap symbol in one or the other of the input strings or the case (iv) when both input strings are empty.

As the algorithm should return the best possible alignment of the input strings, each of the production rules is assigned a scoring function in the scoring algebra. The scoring algebra will take the current score and return an updated score including the weight for the current case (align, delete or insert). An additional function is added to the scoring algebra, the objective function or choice function which defines on how to choose the optimal value.

For each grammar, we can choose a scoring algebra independently. The scoring algebra w is usually based on the arctic or tropical semiring, where $\oplus$ defines the objective function or choice function h and $\otimes$ defines the concatenation of scores and recursive call. Two possible scoring algebras are given in Figure 12.

The signature in ADP provides a set of function symbols that provide the glue between the grammar and the algebra. Thus, a type signature is defined for each of the scoring functions, analogously to type signatures found in functional programming. Figure 13 shows the signature for the Needleman-Wunsch algorithm. This signature fits to both scoring algebras in Figure 12 and the grammar in Equation 2.10. Here, Σ is the input alphabet, thus the input has to be an element of Σ. R is the semiring defining the scores as above.

$$\begin{aligned}
\text{align} &: R \times \left(\tfrac{\Sigma}{\Sigma}\right) \to R \\
\text{deletion} &: R \times \left(\tfrac{\Sigma}{-}\right) \to R \\
\text{insertion} &: R \times \left(\tfrac{-}{\Sigma}\right) \to R \\
\text{empty} &: \epsilon \to R \\
\text{choice} &: [R] \to R
\end{aligned}$$

Figure 13: Signature for the problem of string alignment. The signature lists the situations that need to be evaluated differently by specifying the input and output for each situation. Here, the scores are defined over the semiring R, Σ is the input alphabet, thus elements have to be a letter from Σ, $-$ is the gap symbol and ϵ the empty string.

The last part needed for ADP is the memoization to store the subsolutions. For the Needleman-Wunsch algorithm one simply uses a matrix of size $O(s * t)$ where s and t are the lengths of the input strings to store the subsolutions.

A definition of dynamic programming based on ADP is given in Miklós (2019) together with more detailed descriptions on ADP in general.

`ADPfusion`

Algebraic dynamic programming (Giegerich and Meyer, 2002) in its original form was designed for single-sequence problems. Alignment-style problems were solved by concatenating the two inputs and including a separator. Originally, `ADPfusion` (Höner zu Siederdissen, 2012) was designed along the lines of ADP, but gained provisions to abstract over both, the input type(s), and the index space. With the advent of generalized algebraic dynamic programming (Höner zu Siederdissen et al., 2015a; Höner zu Siederdissen et al., 2015b; Riechert et al., 2016) inputs were generalized to strings and sets, as well as multiple tapes (or inputs) of those. A further generalization to tree and forest structures is described in Chapter 3.

The `ADPfusion` framework is written in the functional language Haskell. Here, it is possible to lift two-tape algorithms to their multi-tape version for regular grammars (Höner

zu Siederdissen et al., 2015a) and automatically deduce the outside (or backward) grammar based on the inside (or forward) version (Höner zu Siederdissen et al., 2015b; Riechert et al., 2016). Beyond many implementations of the Needleman-Wunsch algorithm, the implementation based on `ADPfusion` (Algebraic Dynamic Programming with compile-time fusion of grammar and algebra) (Höner zu Siederdissen, 2012) is designed in a way to be extendable to different scoring functions, problem descriptions, and data structures (Höner zu Siederdissen et al., 2015a).

Inside-Outside algorithms in `ADPfusion`

As the description of algorithms in `ADPfusion` is based on formal grammars, it uses distinct non-terminals for the inside version of the corresponding grammar and its outside version (Höner zu Siederdissen et al., 2015b). Here, all grammars that do not use the outside or backward algorithm are based on the inside grammar and are explained and shown during former sections, e.g. Subsection 2.2.3 and Subsection 2.2.4. Non-terminals of the outside grammar are denoted by adding an asterisk '*'. Non-terminals in a grammar correspond to a certain range of indices of the input structure. Inside non-terminals recurse inside of this range, outside non-terminals on the indices outside of this range. Each inside rule has corresponding outside rule(s). Figure 14 shows an example for an inside-outside rule of a regular grammar (usually called forward-backward) and a context-free grammar (CFG). In the regular case, inside and outside grammar are equivalent except that the indices range beneath the non-terminals differs. The inside algorithm starts from the empty input, the outside algorithm from the complete input as initialization step. Outside grammars are more complicated for CFGs. Thus, given the rule $M \to NO$ from the matrix-chain-multiplication algorithm (see Subsection 2.2.4), the non-terminal M is rewritten into non-terminal N and O, thus it recurses on two parts of the current input. The original rule only uses M as non-terminal, however, to better distinguish the cases, N and O replace M non-terminals. The rule $M \to NO$ calculates a score for splitting up M into N and O. If one wants to know a score for all possible variations of the algorithm instance on N, one calculates its outside value N^*, thus everything outside of this rule. This value is the sum of the outside probability of its predecessor, thus M^* and the inside value for O, as O is part of the rule $M \to NO$, thus we keep the inside value. The outside rule is then $N^* \to M^*O$. This is also shown in Figure 14. One calculates the outside values for O^* analogously, thus $O^* \to M^*N$. Chapter 3 will show inside and corresponding outside grammars for DP algorithms on tree and forest structures.

A more detailed description of the ADP version of tree and forest alignment and editing including the grammars for inside and outside algorithms is given in Chapter 3. Applications of the `ADPfusion` framework can be found in, e.g., Berkemer et al. (2017a), Velandia-Huerto et al. (2016), and Prohaska et al. (2018). An extension to the usual Needleman-Wunsch alignment based on the `ADPfusion` framework is described in Chapter 6.

2.3 Dynamic Programming Algorithms in Bioinformatics

Frequent tasks in bioinformatics include analysis of biological sequences, mainly comparison of different sequences using alignments or analysis of molecule structures which can be

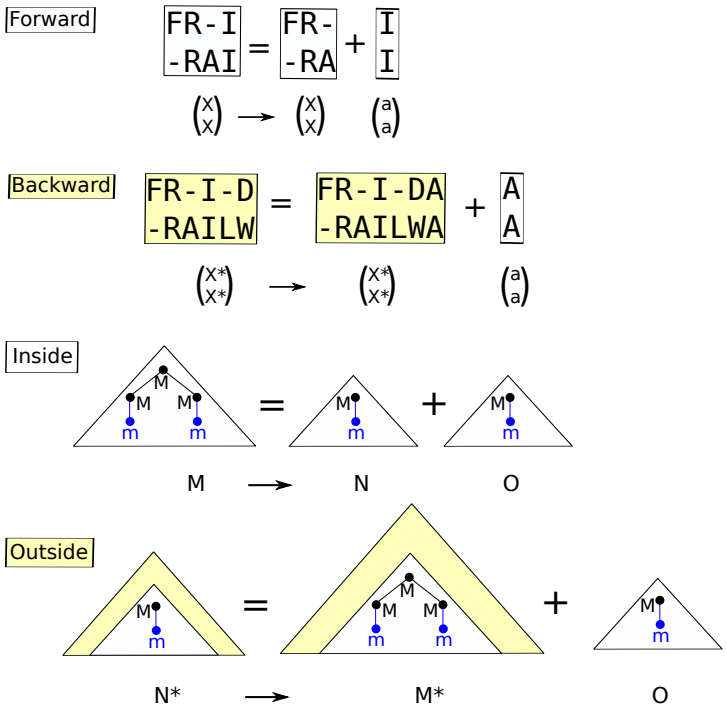


Figure 14: Example of inside and outside versions of a regular two-tape grammar (top, forward and backward) and a single-tape CFG (bottom, inside and outside). Descriptions of corresponding formal grammars are given below each case.

calculated using structure prediction algorithms. Biological sequences include DNA, RNA and proteins (amino acid sequences). More details and biological background is given in Chapter 5. For now, we think of biological sequences as strings of different letters.

Two well-known and basic DP algorithms frequently used in various bioinformatics applications are described in the following subsections. The first will deal with sequence alignment algorithms for which basics were already given with the description of the Needleman-Wunsch algorithm, see Subsection 2.1.2. There exists various extensions and versions of this algorithm which are usually described by a regular grammar. The subsequent section describes the basic algorithm for structure prediction. Its input is just one sequence which is predicted to be able to form a certain structure by folding onto itself. The algorithm can be described by a CFG which is an extension to the grammar for matrix-chain-multiplication, given in Equation 2.12.

Further algorithms will be described in Section 5.4 directly referring to the corresponding biological applications.

2.3.1 Sequence Alignment

The literature on alignments is extensive. However, it its concerned almost exclusively with practical algorithms and applications. The alignment problem for two input strings has an elegant recursive solution for rather general cost models and has served as one of the early paradigmatic examples of dynamic programming (Needleman and Wunsch, 1970; Sankoff and Kruskal, 1983). Since these algorithms have only quadratic space and time requirements for simple cost models (Needleman and Wunsch, 1970; Gotoh, 1982), they are of key importance in practical applications. The same recursive structure easily generalizes to alignments of more than two sequences (Carrillo and Lipman, 1988; Lipman et al., 1989) even though the cost models need to be more restrictive to guarantee polynomial-time algorithms (J. Kececioglu and Starrett, 2004). The computational effort for these exact solutions to the alignment problem increases exponentially with the number of sequences, hence only implementations for 3-way (Gotoh, 1986; Konagurthu et al., 2004; Kruspe and Stadler, 2007) and 4-way alignments (Steiner et al., 2011) have gained practical importance. A wide variety of *multiple sequence alignment* algorithms problems (for arbitrary numbers of input sequences) have been shown to be NP-hard (J. D. Kececioglu, 1993; L. Wang and T. Jiang, 1994; Bonizzoni and Della Vedova, 2001; Just, 2001; Elias, 2006) and MAX SNP-hard (Wareham, 1995; Manthey, 2003). The construction of a practical multiple sequence alignment (MSA) therefore relies on heuristic approximations. These fall into several classes (see e.g. Edgar and Batzoglou (2006) and Baichoo and Ouzounis (2017) for reviews):
(1) *Progressive* methods typically compute all pairwise alignments and then use a "guide tree" to determine the order in which these are stepwisely combined into a multiple alignment of all input sequences. The classical example is `ClustalW` (Larkin et al., 2007). The approach can be extended to starting from exact 3-way (Konagurthu et al., 2004; Kruspe and Stadler, 2007) or 4-way alignments (Steiner et al., 2011).
(2) *Iterative* methods start to align small gapless subsequences and then extend and improve the alignment iteratively until the score converges. The iterative approach is often used as a refinement step in combination with different other basic methods.
(3) *Consistency*-based alignments and *consensus* methods start from a collection of partial alignments (often exact pairwise alignments) to obtain candidate matches and extract a multiple alignment using agreements between the input alignments. A paradigmatic example for the combination of consistency-based alignments and the iterative approach is `DIALIGN` (Morgenstern, 1999) using additionally local motifs as anchors.

Most of the successful multiple alignment algorithms in computational biology combine these paradigms. For example `T-COFFEE` (Notredame et al., 2000) and `ProbCons` (Do et al., 2005) use consistency ideas in combination with progressive constructions; `MUSCLE` (Edgar, 2004) and `MAFFT` (Katoh et al., 2005) combine progressive alignments with iterative refinements.

A key assumption underlying consistency based methods is transitivity: considering three input sequences x, y, and z, if x_i aligns with y_j and y_j aligns with z_k, then x_i should also align with z_k. While this property holds for the pairwise constituents of a multiple alignment, it is a well known fact that the three score-optimal alignments that can be constructed from three sequences in general violate transitivity, see Fig. 15. `TRANSALIGN` (Malde and Furmanek, 2013) uses transitivity to align input sequences to a

```
      (a)              (b)              (c)              (d)
A 0000111110000  A 0000111110000  B 000011011----  B 000011011
B 000011011----  C 1000----10000  C 1000----10000  C 100010000
   s = 4            s = 4            s = -4           s = 5
```

Figure 15: Alignments of three binary sequences `A`, `B`, and `C` with a simple scoring model considering additive contributions of columns. In this example, we use a score of $+1$ for matches, 0 for mismatches, and -1 for gaps. Alignment (c) is transitively implied by optimal alignments (a) and (b), but it is not an optimal pairwise alignment of `B` and `C` which is displayed in (d).

target database using an intermediary database of sequences to increase the search space. Here, intermediary sequences show which subsequences of input and target sequence can be transitively aligned. This may result in a few well aligned subsequences that are then extended to one aligned region via a simple scoring function. The same notion of transitivity is also used in `psiblast` (Altschul et al., 1997) to stepwisely increase the set of sequences that are faintly similar to an input sequence.

This thesis includes theoretical work on alignments of trees and forests in Chapter 3 and a more general approach to alignments based on accepted input structures and a generalized view on alignments in Chapter 4. As alignments are the underlying principles of a large number of algorithms in bioinformatics, Chapter 6, Chapter 7 and Chapter 8 make use of alignments applied to biological sequences where Chapter 6 includes an adapted version of the basic alignment algorithm to detect duplicated characters.

2.3.2 Structure Prediction

Biological molecules, especially RNA sequences and proteins, are required to adapt a certain shape in order to be functional. Thus, the sequences fold onto themselves such that they form structure elements that stabilize the molecule and open possibilities to bind to or interact with other molecules or elements of the cell. A more detailed description of the biological processes is given in Chapter 5. Here, we focus on the algorithmic principles as structure prediction (or short 'folding') is based on DP algorithms.

Examples are shown using RNA sequences, however protein folding has similar principles. An RNA molecule is a sequence composed out of four letters (bases), A, C, G and U. The building blocks can form certain connections called base pairs. The set of possible base pairs is $\mathcal{P} = \{AU, UA, CG, GC, UG, GU\}$. Given a sequence $s = s_1, .., s_n$ it has to hold that

- base pairs are formed between two positions in the structure s_i, s_j, $i \neq j$ if $s_i s_j \in \mathcal{P}$,
- every base of the sequence is included in at most one base pair,
- base pairs can be nested thus if s_i, s_j, $i < j$ is a base pair, s_k, s_l with $k < i < j < l$ and $s_k s_l \in \mathcal{P}$ can form a base pair,
- crossing base pairs are not allowed, thus if s_i, s_j, $i < j$ is a base pair, s_k, s_l with $k < i < l < j$ cannot be a base pair,

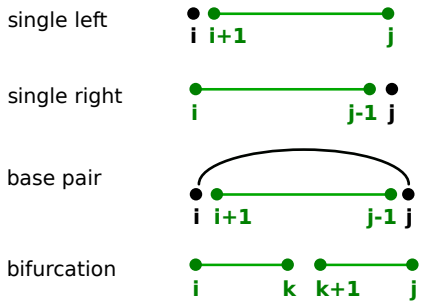


Figure 16: Recursion cases for the Nussinov algorithm. Black circles represent bases, if they are connected by an arc, they form a base pair. Horizontal lines are the input sequence. The green parts depict the input for the next recursion step, black parts show structures corresponding to terminals in the formal grammar. Letter correspond to the indices of the input structure as given in the recursion equation.

- in biological applications, neighboring bases s_i, s_{i+1} cannot form a base pair due to space constraints. Thus, a minimal distance between bases is required to be able to form a base pair.

One of the first algorithms to calculate structures of biological molecules is the *Nussinov algorithm*, a DP algorithm that aims to maximize the possible number of base pairs (Nussinov et al., 1978; Nussinov and Jacobson, 1980).

The recursion equation for the Nussinov algorithm can be written as follows (*cf.* Durbin et al. (1998)):

$$s(i,j) = max \begin{cases} s(i+1,j) + w_{singleleft} \\ s(i,j-1) + w_{singleright} \\ s(i+1,j-1) + w_{basepair} \\ max_{i<k<j} s(i,k) + s(k+1,j) + w_{bifurcation} \end{cases} \tag{2.13}$$

The Nussinov algorithm iterates over the input sequence from the front and from the back or splits the input sequence in two parts and recurses on them. This can also be seen in Figure 16 that depicts the four recursion cases of the Nussinov algorithm. In order to calculate the maximum possible number of base pairs for an input sequence, a DP algorithm can be used. Here, a memoization table is filled, as depicted in Figure 17. The initialization is done by

$$s(i,i-1) = -\infty$$

$$s(i,i) = max(w_{singleleft}, w_{singleright})$$

After the initialization step, the memoization table is filled such that short base pairs, e.g. starting with a range of only one base in between, are calculated first, thus the algorithm is going from short-range to long-range base pairs. This corresponds to filling the matrix diagonal by diagonal starting in the middle going toward the upper corner on the r.h.s. The structure of the solution can be found by backtracking where the base pairs

included in the solution correspond to the cells in the matrix that show an increase of the score, shown by the colored squares in Figure 17.

As the algorithm tries to maximize the number of base pairs, a possible scoring would be to only score $w_{basepair}$ positively, e.g. by 1, and give a neutral score to the other cases, thus 0. This scoring scheme is used in Figure 17. To simplify the grammar representation, nucleotides will be denoted by n and nucleotides forming a base pair will be denoted $'n$ and n'. In a different setting, n is denoted by '.' and $'n$ and n' as (and), respectively. This is called the dot-bracket notation and will be explained further in Chapter 5.

The Nussinov algorithm can be described by a context-free grammar (CFG). The version of the grammar shown in Equation 2.14 is written analogously to the recursion equations (Equation 2.13), however, there exist multiple ways to write down the Nussinov algorithm.

$$S \rightarrow \underbrace{Sn}_{singleleft} \mid \underbrace{nS}_{singleright} \mid \underbrace{'nSn'}_{basepair} \mid \underbrace{SS}_{bifurcation} \mid \underbrace{\epsilon}_{empty} \tag{2.14}$$

Again, one can draw a parse tree matching the derivations of the grammar. Figure 18 displays the parse trees of the solutions in Figure 17 where the red solution matches the black and red parse tree and the solution depicted in blue belongs to the parse tree drawn in black and blue. This way of depicting a structure is commonly used in bioinformatics when comparing distinct structures. More details on how the comparison can be done is given in Chapter 3.

The Nussinov algorithm only focuses on optimizing the number of possible base pairs and completely omits biochemical or thermodynamical properties of the molecules. Zuker

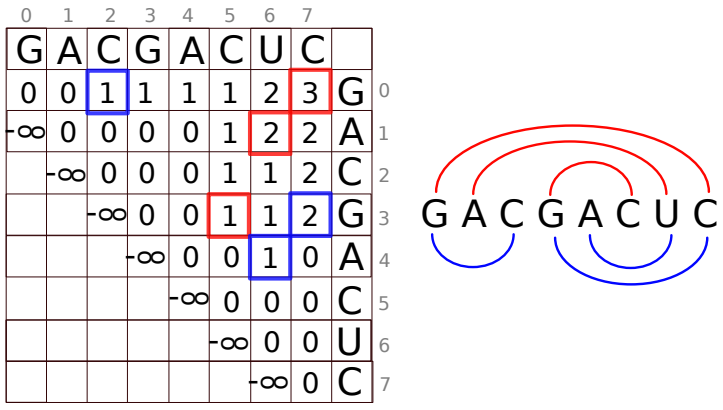


Figure 17: Example of the Nussinov algorithm calculating the maximum number of possible base pairs. A score for all possible base pair formations is calculated starting from low range interactions $(i, i+1)$ going towards long range interactions $(i, i+n-1)$. The red and blue squares show co-optimal solutions with 3 base pairs, respectively. The solutions are depicted on the right. The scores are 1 for a possible base pair and 0 otherwise. The objective function maximizes the current value.

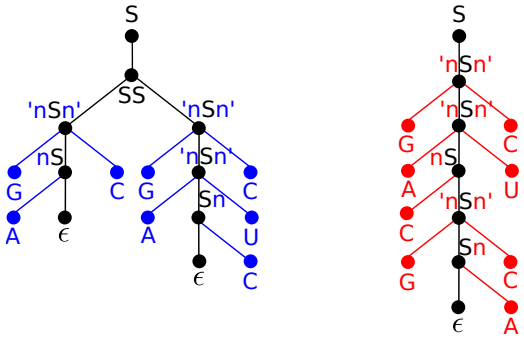


Figure 18: Parse trees corresponding to the solutions of the Nussinov algorithm on the sequence *GACGACUC* as given in Figure 17. The inner nodes depict applied grammar rules as given in Equation 2.14. The leaves of the trees are the letters of the input sequence such that the input can be recovered by traversing the tree in preorder, thus starting at the root and always visiting the leftmost child first. Red and blue color match the structures depicted in Figure 17.

and Stiegler (1981) were one of the first to formulate the folding algorithm based on thermodynamical rules by setting energy parameters for different conformations of the sequence. The recursion equations of the *Zuker algorithm* are depicted in detail in Zuker and Stiegler (1981) and Bompfünewerer et al. (2008).

CHAPTER 3

Dynamic Programming on Trees and Forests

Contents

This chapter will deal with dynamic programming algorithms on tree and forest structures as inputs. Trees as data structures sit "somehow between" strings and sets. They have more complex structures, in that each node not only has siblings, but also children compared to lists which only have siblings, i.e., one kind of adjacency, for each node. Sets on the other hand are typically used based on the principle of having edges between all nodes, with each edge having a particular weight. This alone makes trees an interesting structure to design a formal dynamic programming environment for. Tree and forest structures have important functions in computer science, bioinformatics and linguistics as their structure allows to represent nested data structures such as parse trees of programming code or natural languages. Further applications in bioinformatics will be given in Chapter 5 and Chapter 7.

The aim of the present chapter is to develop a general framework as a generic basis for the design and implementations of dynamic programming (DP) algorithms on trees and forests. Here, the focus is on well-known algorithms with existing implementations as examples because in these cases the advantages of the algebraic dynamic programming (ADP) approach (as described in Subsection 2.2.7) can be discussed directly in comparison to the literature. Since DP algorithms are usually formulated in terms of recursion equations, their structure is nicely reflected by formal grammars. These are more compact than the recursions and more easily lend themselves to direct comparisons. We give a small set of combinators – functions that formalize the recursive deconstruction of a given tree structure into smaller components – for the design of such formal languages. This chapter is based on Berkemer et al. (2017b) with the title *Algebraic Dynamic Programming on Trees and Forests*. Additional information is given in the original work and in Appendix A.

3.1 Trees and Forests as Data Structures

Trees and forests as data structures are uniquely defined based on two partial orders of the elements: the ancestor order and the sibling order (see also Subsection 2.2.1). Thus, elements of trees or forests still exhibit a well defined structure that leads to two obvious total orders on the elements, the preorder and postorder on trees and forests as defined in Subsection 2.2.1. This chapter will also deal with the alignment of trees and forests, however, a more generalized way for the construction of alignments will be given in Chapter 4. In order to define formal grammars for DP algorithms on trees and forests, we have to define how to split a tree or a forest into its substructures. On strings, the concatenation operator is assumed implicitly to concatenate single symbols of the string. We make the operators explicit as there is more than one operator on trees and forests. Each operation transforms a tree or forest on the left-hand side of a rule ($\rightarrow$) into one or two elements on the right-hand side of a rule. Thus, the operators defined below will be used in production rules of formal grammars, defined in Subsection 2.2.3.

Let T be a rooted, ordered tree and let F be the ordered forest of trees $T_1, \ldots, T_n$, with root nodes $r_1, \ldots, r_n$ as defined in Definition 12 on page 25. Labels are associated with roots, leaves and inner nodes of the tree or forest. For each tree T it holds that it is empty and contains no nodes, or there is a designated node r, the root of the tree. We now define the set of operations that will lead to dynamic programming operations on ordered trees and forests. The operations are shown within an example in Figure 19.

Definition 22 (Empty Tree or Forest)
If T is empty, then the rule $T \to \epsilon$ yields ϵ. If F is empty, then the rule $F \to \epsilon$ yields ϵ.

Definition 23 (Single Root)
If T with root r contains just the root r, then $T \to r$ yields r.

Definition 24 (Root Separation)
Given non-empty T, the rule $T \to r \downarrow F$ yields the root r of T, as well as the ordered forest $F = T_{r,1} \dots T_{r,n}$ of trees. Each $T_{r,k}$ is uniquely defined as the k'th child of the root r of T. $T \to F \downarrow r$ is isomorphic to this rule.

Definition 25 (Leaf Separation)
Given non-empty T, the left-most leaf $l = lleaf(T)$ that is defined as the leaf the can be reached by following the unique path from the root r of T via each left-most child until a leaf has been reached. We write $T' = T - l$ for the tree obtained from T by deleting the leaf l and its incident edge. The rule $T \to l \lrcorner T'$ breaks up T into the leaf l and the remaining tree $T' = T - l$.
The rule $T \to T \llcorner s$ likewise extracts the right-most leaf.

Definition 26 (Forest Root Separation)
Let $F = [T_1, \dots, T_k]$ be a non-empty forest and let r be the root of T_k, the right-most tree in F. The rule $F \to F \downarrow r$ separates the right-most root *r, that is, the resulting forest $F = [T_1, \dots, T_{k-1}, T_{c1}, \dots, T_{cm}]$ replaces T_k in the the original forest by the child-trees $T_{c1}, \dots, T_{cm}$ of r. The rule $F \to r \downarrow F$ analogously* separates the left-most root *r by replacing the the leftmost tree of F by the child-trees of its root.*

Definition 24 is a special case where the forest only consists of one tree.

Definition 27 (Forest Separation)
The rule $F \to T \circ F'$ separates the non-empty forest F into the left-most tree T, and the remaining forest F'. T is uniquely defined by the ordering of F.
The rule $F \to F' \circ T$ likewise yields the right-most tree.

The following lemma gives that both, tree and forest decomposition are well defined and always lead to a decomposition using at least one of the definitions above.

Lemma 1 (Tree Decomposition)
Given any (empty or non-empty) tree T, at least one of the rules from Definition 22 to Definition 25 yields a decomposition of the tree, where either T is decomposed into "atomic" elements (ϵ or a single node), or into two parts, each with fewer vertices than T.

Proof 1 *Every tree T is either empty (Definition 22), contains just one node which then is the root (Definition 23), or contains one or more nodes (Definition 24 and Definition 25), of which one is separated from the remainder of the structure.*

Lemma 2 (Forest Decomposition)
For every forest F the operations Definition 22, Definition 26 or Definition 27 yield a decomposition.

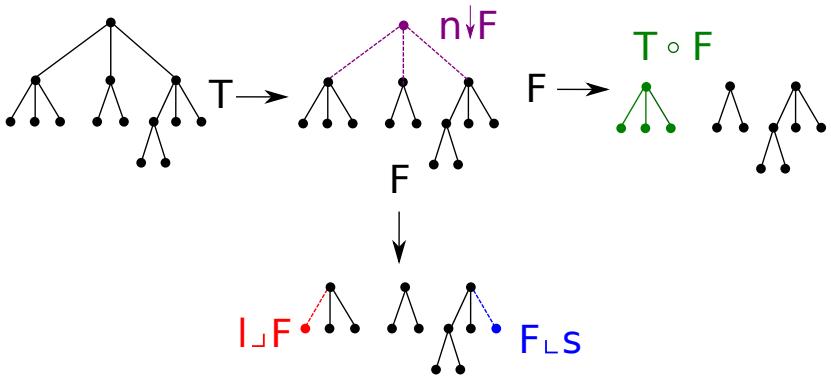


Figure 19: Examples for operations defined on a tree or forest. Rules depicted here correspond to root separation in a tree or forest (purple), forest separation into the leftmost tree (green) and the remaining forest and leaf separation of the leftmost (red) and rightmost (blue) leaf. Dotted lines indicate edges that are to be removed when the corresponding node is split from the tree.

Proof 2 *The argument is analogously to the case of trees. The forest root separation cases given in Definition 26 remove the left- or right-most root from the forest, hence they constitute a decomposition of the structure. This might lead to a larger number of trees in the resulting forest but reduces the total number of nodes.*

Given those tree and forest concatenation operators, we are now able to define formal grammars to describe DP algorithms on trees and forests.

Lemma 3 (Totality)
Given an arbitrary tree or forest, at least one of the operations specified in Definition 22 up to Definition 27 and Lemma 1 and Lemma 2 can be applied. Each of these operations reduces the number of nodes associated with each non-terminal symbol or results in a single node or an empty structure. Thus any finite input tree or forest can be decomposed by a finite number of operations.

Proof 3 *Let F be a forest. For the case of the empty forest, we apply Definition 22. In the case of more than one tree, we apply Definition 27 which yields a single tree and another forest that are both smaller than the original forest. This is also shown in Lemma 2.*

In case of Definition 27, the resulting forest structure can be empty such that the original forest only consisted of a single tree. Here, we continue with the possible tree structures.

A tree T can be empty, or consist either of a single node or more than one node. For the empty case, we apply Definition 22. In case of the tree consisting only of a single node, we apply Definition 23. If T consists of more than one node, we either apply Definition 24 or Definition 25. In case of Definition 24 the resulting structure consists of a single node (an atomic structure) and a forest which in total has a smaller number of nodes than the original tree. In the case of Definition 25, one leaf is eliminated from the tree, thus this yields a tree that just differs from the original structure by one leaf. For the case of

algorithms working on two input structures, it is ensured that at least one of the current structures is decomposed such that the total number of nodes decreases within the next recursion step.

The operators on tree and forest structures defined above specify how these data structures can be traversed. They specify a basis for the formulation of at least a large collection of DP problems using tree and forest decomposition. Different DP problems thus can be formulated (as a formal grammar) based on the same set of decomposition rules. There are many ways of designing DP algorithms on tree structures (Chen, 2001; Schwarz et al., 2017; Tai, 1979). At present we lack a formalized theory of decompositions for arbitrary data structures hence we cannot formally prove that this set of operators is sufficient to formulate any meaningful DP problem on trees. If necessary, however, additional decomposition operators can be added to the list.

3.2 Single-tape DP on Trees and Forests

Dynamic programming algorithms on a single input tend to be concerned with the internal structure of the input. Single-tape problems and their associated grammars therefore deal with the structural decomposition of this one input. A well-known example in bioinformatics is the Nussinov algorithm (Nussinov and Jacobson, 1980), described in Subsection 2.3.2 that is applied to one input sequence and returns the optimal structure based on a scoring scheme (often maximal number of paired nucleotides).

Analogously to single-tape DP on strings, there exist single-tape DP algorithms on trees and forests. Here, problems include the search for an optimal partitioning of the tree or detection of paths within the input structure. In contrast, two- or multi-tape DP algorithms are usually used to compare two or more input structures with each other with the aim of finding an optimal way of transforming one into the other or composing them into a consensus structure. We start with two examples of DP algorithms on single trees. Both solve well-known problems in computational biology but are also easy to state without reference to their usual applications. We consider these trees as ordered since in practice they are given with fixed, albeit arbitrary, order of the children. In the following section we will then address algorithms that take pairs of labeled trees as input.

3.2.1 The Minimum Evolution Problem

Given a tree T with leaf set $\mathcal{L}$, vertex set $V(T)$ and edge set $E(T)$. The relation $chd(v)$ returns the set of children of a vertex v in a tree. Σ is the alphabet of the vertex labels. The problem is described as given a labelling $\ell : \mathcal{L} \to \Sigma$, and a similarity function $w : \Sigma \times \Sigma \to \mathbb{R}$, find an extension $\tilde{\ell} : V(T) \to \Sigma$ such that: (i) $\tilde{\ell}(v) = \ell(v)$ on $\mathcal{L}$ and (ii) $w(\ell) := \sum_{\{u,v\} \in E(T)} w(u, v)$ is maximal. Maximizing similarity amounts to minimizing the number of evolutionary events that occur along the edges of the (phylogenetic) tree, explaining the traditional name of the problem. This combinatorial optimization problem is known as the "Minimum Evolution" or "Small Parsimony" problem.

A well-known solution to the Minimum Evolution Problem, known as Sankoff's Algorithm (Sankoff, 1975), consists of computing for each complete subtree of T, which by construction is rooted in a vertex $v \in V$, and each possible label $a \in \Sigma$ of v, the score of

the best scoring labeling of that subtree. Let us call this quantity $S_v(a)$. It satisfies the recursion

$$\begin{aligned} S_v(a) &= \begin{cases} 0 & \text{if } \ell(v) = a \\ -\infty & \text{otherwise} \end{cases} \qquad \text{for all } v \in X \\ S_v(a) &= \sum_{c \in \mathrm{chd}(v)} \max_{\ell(c)} \left(S_c(\ell(c)) + w(a, \ell(c)) \right) \qquad \text{for all } v \in V(T) \setminus \mathcal{L} \end{aligned} \tag{3.1}$$

The initialization for $v \in \mathcal{L}$ enforces that a score different from $-\infty$ is obtained only for configurations where the label a at leaf v coincides with the input $\ell(v)$. Otherwise, v obtains a score of $-\infty$ that propagates upwards through the recursion for the inner nodes. Probabilistic versions of the algorithm assign probabilities proportional to $\exp(score)$ to each configuration, and thus a probability of 0 to any solution in which the leaf labels deviate from the input. The grammar to Sankoff's algorithm, Equation 3.1, is given by

$$T \to x \downarrow F \qquad\qquad F \to T \circ F \;\Big|\; \epsilon \tag{3.2}$$

Sankoff's algorithm, and many others often used in particular in computational biology, proceeds using two decomposition steps, namely, the separation of the root from the forest of its child trees, and the stepwise decomposition of a forest into its component trees.

We remark that the same grammar can also be use to describe Fitch's solution to the Small Parsimony problem (Fitch, 1971) for binary trees, which was later generalized by Hartigan (Hartigan, 1973) for arbitrary trees. Even though the same grammar is used, the scoring algebra is completely different in the two algorithms, see Figure 20 and Figure 21. Each scoring algebra consists of functions corresponding to the rules in the grammar, thus in this case, a `tree` function, a `forest` function, a function for the `empty` case together with the `choice` function form the algebra. Each (sub)solution consists of a list of pairs (s_T, V_T) consisting of the score s_T for the input tree T and a set V_T, containing the nodes included in the solution. Thus, each instance of the problem will also receive the list of pairs $[(s_T, V_T)]$ as an input. Let A be the set of possible labels in the tree and l the length of the input list. Then Fitch/Hartigan uses the scoring algebra depicted in Figure 21. See also Chapter 7 for a more detailed description and application of the Fitch algorithm. For the Sankoff version, only a label-dependent score w for the trees is given as input, with L_T as the set of labels in the current subtree T, as it can be seen in Figure 20. Thus, this algebra (Figure 20) considers the Sankoff version of the small parsimony problem.

$$\begin{aligned} \mathrm{forest}([s], F) &= ([s] + +s_F, F') \\ \mathrm{tree}([s], T) &= ([s], w(L_T)) \\ \mathrm{empty}([s], \epsilon) &= 0 \\ \mathrm{choice}([s_1, \ldots, s_k]) &= s_m \text{ with } s_m = max_i(s_i) \end{aligned}$$

Figure 20: Scoring algebra for the Sankoff version of the small parsimony problem. Here, $[s]$ is the list of scores for possible (sub)solutions and with ++ a further element is added to the list.

$$
\begin{aligned}
\text{forest}([(s_T, V_T)], F) &= ([(s_T, V_T)] + +(s_{T'}, V_{T'}), F') \\
\text{tree}([(s_T, V_T)], T) &= (s_{T'}, V_{T'}) \text{ where} \\
&\quad k(a) =\mid \{T' \mid a \in V_{T'}\} \forall a \in A, \\
&\quad s = \sum_{T'} s_{T'}, \\
&\quad k = max_a(k(a)), \\
&\quad V_{T'} = V_{\{x\}} \cup \{a \mid k(a) = k\}, \\
&\quad s_{T'} = s + l - k. \\
\text{empty}([(s_T, V_T)], \epsilon) &= (0, 0) \\
\text{choice}[(s_T, V_T)_1, \ldots, (s_T, V_T)_k] &= (s'_T, V'_T)_i \text{ with } s'_T = max_{s_T, i}((s_T, V_T)_i)
\end{aligned}
$$

Figure 21: Scoring algebra for the Fitch/Hartigan version of the small parsimony problem.

3.2.2 The Phylogenetic Targeting Problem

Here we are again given a tree T with leaf set $\mathcal{L}$, this time together with a weight function $w : \mathcal{L} \times \mathcal{L} \to \mathbb{R}$. The weight w models the amount of information that can be gained by comparing data measured for a pair of two taxa, i.e., a pair of leaves. Two pairs of taxa u, v and x, y are said to be *phylogenetically independent* if the two paths from u to v and from x to y, respectively, have no edge in common (Maddison, 2000). The task is to find a set $\mathbb{S}$ of mutually phylogenetically independent pairs of taxa that maximizes the total amount of information, i.e., that maximizes the total score $f(\mathbb{S}) = \sum_{\{x,y\} \in \mathbb{S}} w(x, y)$, see also Figure 22 for an example. This problem is of practical interest whenever the acquisition of the data is very expensive, e.g., when extensive behavioral studies of animals need to be conducted. It is natural then to "target" the most informative selection of species (Maddison, 2000; Arnold and Nunn, 2010).

We consider here only binary, i.e., fully resolved phylogenetic trees. Feasible solutions to this problem thus consist of a tree T endowed with a set of disjoint paths that connect pairs of leaves. Let us now consider a subtree. Depending on the choice of the paths, we can distinguish two distinct types of partial solutions: In subtrees of type U all paths between leaves in U are confined to U. In the other case, which we denote by W, a path leaves the subtree through its root. In this case the root is connected by a path to one leaf in the subtree. This path is incomplete and must be connected to a leaf in another subtree. The solution of the complete problem must be of type U. The child-trees of a type U tree are either both of type U or both of type W. In the latter case a path runs through the root of U and connects the roots of the two type-W children. Further, denote by G a forest consisting of two trees of the same type, and let H be a forest comprising

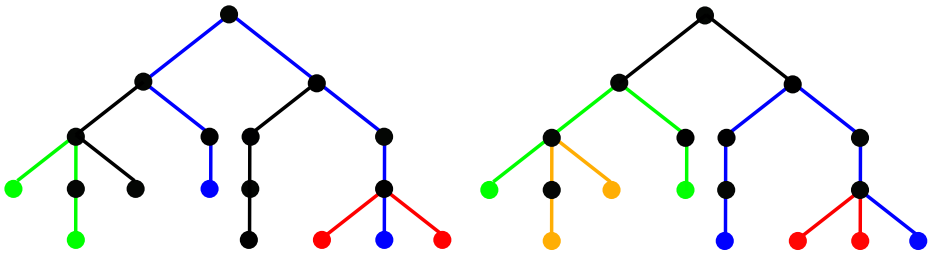


Figure 22: Two possible systems depicting edge disjoint paths between leaves of a tree. Paths are shown in different colors and do not have to be vertex disjoint. Start and end nodes of the paths are shown in the corresponding color. Unused edges and nodes as well as inner nodes are shown in black.

two trees of different type. This yields the following grammar:

$$\begin{aligned} U &\to r \downarrow G \\ G &\to U \circ U \mid W \circ W \mid \epsilon \\ W &\to r \downarrow H \\ H &\to U \circ W \mid W \circ U \mid \epsilon \end{aligned} \tag{3.3}$$

The first and third rule, respectively, remove the root of a binary tree and leave us with the forest of its two children. The rules for G and H describe the composition of these two forests in terms of its constituent trees. Note that $U \circ W$ and $W \circ U$ refer to distinct solutions since we assume that the phylogenetic tree is given as an ordered tree. There are again several possible scoring algebras. As described in Arnold and Nunn (2010) and Arnold and Stadler (2010), to obtain the best possible path system, the objective function maximizes over the best possible path systems of the subtrees given from the current instance of the program and adds up the scores.

3.3 Two-tape and Multi-Tape DP on Trees and Forests

Even though a forest is composed out of a sequence of trees, a forest F is considered as a single input. Additionally, any forest can be represented as a tree with the introduction of a root node r_0 of which the trees of the forest are the children. As such, a definition on trees is enough to cover forests as well. See Chauve et al. (2016) for a similar definition. As examples for DP algorithms on trees and forests, we will use the following sections to describe two important and often-used DP problems on trees, the tree alignment (T. Jiang et al., 1995) (Section 3.4) and tree editing (Selkow, 1977; Tai, 1979) (Section 3.5) problems. Tree editing is concerned with finding the optimal edit script that transforms the first input tree into the second input tree. Tree alignment, on the other hand, gives the optimal alignment of two trees with each other, as will be described below in more detail. Figure 23 shows the differences between tree alignment and tree editing of two trees F_1 and F_2. Tree alignment will find an optimal consensus tree (top) whereas tree editing will find the least number of edit operations in order to transform one tree into the other

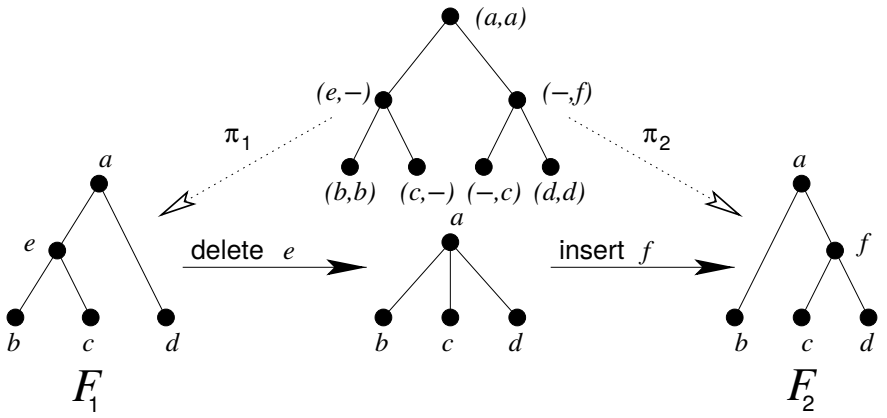


Figure 23: Alignment of two forests F_1 and F_2 (top) and a sequence of edit operations (relabel, delete, insert) in order to transform F_1 into F_2 (below). Tree alignment conserves the structures of both original trees that can be recovered using the mapping π_1 or π_2, respectively. For more details, see Sec. 3.4.

(bottom). Hence, tree alignment conserves the original structure of both trees opposite to tree editing. Regarding string editing and string alignment, one can see analogous cases where editing results in the largest common substructure and alignment in the smallest common superstructure of its inputs. More details will be explained in Section 3.4 and Section 3.5, respectively. These algorithms serve as a tutorial on how grammars on trees are to be formulated. We expand on earlier work by introducing several variants. While some are known from previous work (Höner zu Siederdissen et al., 2015a; Riechert et al., 2016; Höner zu Siederdissen et al., 2015b), the terse and high level notation, ability to construct combined grammars, and automatic derivation of the corresponding outside grammar give a unique framework.

3.4 Tree Alignment

Our grammar for the case of tree alignment is based on the grammars formulated in Schirmer (2011) and Schirmer and Giegerich (2011). We expand those grammars with explicit tree concatenation operators as shown in the previous section. This allows our algorithms to parse both trees and forests as input structures.

Furthermore, we extend the grammars with the automatic derivation of an outside algorithm (Höner zu Siederdissen et al., 2015b), described in Subsection 3.4.3. The combination of the inside and outside grammar allows for easy calculation of match probabilities. Since this is automatic, any user-defined grammar can be extended in this way as well. Though we point out that this still requires careful design of the inside grammar (Chauve et al., 2016). Representations of alignments (*cf.* Figure 23) are typically in the form of a tree that includes nodes from both trees, where matching nodes are given as a pair and deleted nodes from both trees are inserted in a partial-order preserving

fashion.

Definition 28 (Tree Alignment)
Consider a forest W with vertex labels taken from $(\Sigma \cup \{-\}) \times (\Sigma \cup \{-\})$ with Σ as alphabet. Then we obtain mappings $\pi_1(W)$ and $\pi_2(W)$ by considering only the first or the second coordinate of the labels, respectively, and by then deleting all nodes that are labeled with the gap character '$-$', see Figure 23 above and Definition 2 on page 17. W is an alignment of the two forests F_1 and F_2 if $F_1 = \pi_1(W)$ and $F_2 = \pi_2(W)$ and can also be denoted as a forest alignment (F_1, F_2).

The cost of the alignment (F_1, F_2) based on the original forests F_1 and F_2 is the sum of the costs based on the labels in (F_1, F_2) as formulated in Equation 3.4. Each label consists of a pair of labels (v_1, v_2), whereas v_1 corresponds to a label in the vertex set of F_1 or a gap symbol '$-$' and analogously for v_2.

$$\gamma((F_1, F_2)) = \sum_{(v_1,v_2)\in(F_1,F_2)} \gamma_{v_2}^{v_1} \tag{3.4}$$

Every alignment $W = (F_1, F_2)$ defines unique mappings $\pi_1(W)$ and $\pi_2(W)$, but the converse is not true. The minimum cost alignment is in general more costly than the minimum cost edit script. We will need a bit of notation. Let F be an ordered forest. By $i : F$ we denote the subforest consisting of the first i trees, while $F : j$ denotes the subforest starting with the $j + 1$-st tree. By $F^{\downarrow}$ we denote the forest consisting of the children-trees of the root $v = r_F$ of the first tree in F. $F^{\rightarrow} = F : 1$ is the forest of the right sibling trees of F.

Now consider an alignment A of two forests F_1 and F_2. Let $a = r_A$ be the root of its first tree. We have either:

i) $a = (v_1, v_2)$. Then $v_1 = r_{F_1}$ and $v_2 = r_{F_2}$; $A^{\downarrow}$ is an alignment of $F_1^{\downarrow}$ and $F_2^{\downarrow}$; $A^{\rightarrow}$ is an alignment of $F_1^{\rightarrow}$ and $F_2^{\rightarrow}$.

ii) $a = (v_1, -)$. Then $v_1 = r_{F_1}$; for some k, $A^{\downarrow}$ is an alignment of $F_1^{\downarrow}$ and $k : F_2$, and $A^{\rightarrow}$ is an alignment of $F_1^{\rightarrow}$ with $F_2 : k$.

iii) $a = (-, v_2)$. Then $v_2 = r_{F_2}$; for some k, $A^{\downarrow}$ is an alignment of $k : F_1$ and $F_2^{\downarrow}$ and $A^{\rightarrow}$ is an alignment of $F_1 : k$ with $F_2^{\rightarrow}$.

These three cases imply the following dynamic programming recursion:

$$S(F_1, F_2) = \min \begin{cases} S(F_1^{\downarrow}, F_2^{\downarrow}) + S(F_1^{\rightarrow}, F_2^{\rightarrow}) + \gamma_{v_2}^{v_1} \\ \min_k S(F_1^{\downarrow}, k : F_2) + S(F_1^{\rightarrow}, F_2 : k) + \gamma_{\varnothing}^{r_{F_1}} \\ \min_k S(k : F_1, F_2^{\downarrow}) + S(F_1 : k, F_2^{\rightarrow}) + \gamma_{r_{F_2}}^{\varnothing} \end{cases} \tag{3.5}$$

with initial condition $S(\emptyset, \emptyset) = 0$. The formal grammar underlying this recursion is

$$\begin{array}{lcl|c|l} \binom{F}{F} & \rightarrow & \binom{T}{T} \circ \binom{F}{F} & \epsilon & \\ \binom{T}{T} & \rightarrow & \binom{n}{n} \downarrow \binom{F}{F} & \binom{-}{n} \downarrow \binom{F}{F} & \binom{n}{-} \downarrow \binom{F}{F} \end{array} \tag{3.6}$$

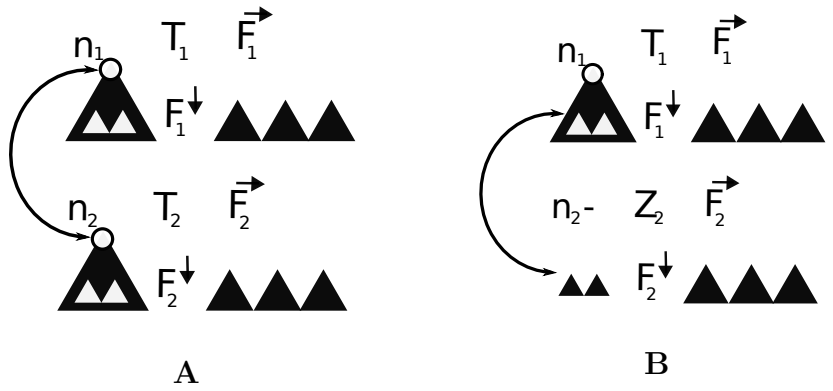


Figure 24: Alignment of two forests 1(top) and 2(bottom). Each of the forests is split into its left-most tree T_1, T_2 and the remaining forest $F_1^{\rightarrow}, F_2^{\rightarrow}$. **A** The left-most tree will be split into its root nodes n_1, n_2 and the forests of its children, $F_1^{\downarrow}, F_2^{\downarrow}$. **B** In case n_2 is deleted, a gap symbol Z_2 is added and the recursion continues with $F_1^{\downarrow}$ and $F_2^{\downarrow}$.

It is worth noting that single tape projections of the form $T \rightarrow - \downarrow F$ make perfect sense. Since $-$ is a parser that always matches and returns an empty string, which in turn is the neutral element of the concatenator $\downarrow$ this formal production is equivalent to $T \rightarrow F$. Hence, it produces a forest F that happens to consist just of a single tree T.

As depicted in Figure 24, fixing the left-most tree T in a forest, there are two directions to traverse the forest: downwards towards the forests of the root's children $F^{\downarrow}$ and sideways towards the remaining forest $F^{\rightarrow}$. Regarding one single forest, the subforests $F^{\downarrow}$ and $F^{\rightarrow}$ are two disjoint entities, thus once split, they do not share any nodes in a further step of the decomposition algorithm. The grammar for tree alignment as described above is inefficient, however, because there is no explicit split between $F^{\downarrow}$ and $F^{\rightarrow}$ in the first step. The grammar shown in Equation 3.7 explicitly splits the two forests in an earlier step to avoid redundancy.

An efficient variant that makes use of a number of facts that turn this problem into the equivalent of a linear grammar on trees has been described in Höchsmann (2005). Trees are separated from their forests from left to right, and forests are always right-maximal. Given a local root node for the tree, and a ternary identifier (`{T,F,E}`), each forest, tree, and empty forest can be uniquely identified. Trees by the local root, forests by the local root of their left-most tree, and empty forests by the leaves "above" them. The asymptotic running time and space complexity for trees with m and n nodes respectively is then $O(mn)$. This also holds for multifurcating trees as the decomposition rules do not care if the tree is a binary tree or not. The children of a node in a tree always form a forest, independent of the number of children.

If the nodes of the tree are labelled in preorder fashion several operations on the forest can be done more efficiently. By splitting a forest into a tree and the remaining forest, we need to store the tree's indices to know where it is located in the forest. In case of a pre-order indexing we can take the left-most tree without storing additional indices as the left border of the tree is the smallest index (which is its root) of the original forest and the

right border of the tree is just the predecessor of the left-most root of the remaining forest. Thus, storing the roots' indices in a forest will directly give us the right-most leaves of the corresponding trees.

We finally consider a variant of Equation 3.6 that distinguishes the match rule $\binom{T}{T} \to \binom{n}{n}\binom{F}{F}$ with a unique non-terminal $\binom{T}{T}$ on the left-hand side of Equation 3.7. This rule, which corresponds to the matching of the roots of two subtrees, is critical for the calculation of *match probabilities* and will play a major role in Subsection 3.4.3. The non-terminals $\binom{T}{Z}$ and $\binom{Z}{T}$ designate insertion and deletion states, respectively. Thus, we can formulate the formal grammar for tree alignment as follows:

$$\begin{array}{lll} \text{iter} & \binom{F}{F} & \to \binom{T}{T} \circ \binom{F}{F} \mid \binom{Z}{T} \circ \binom{F}{F} \mid \binom{T}{Z} \circ \binom{F}{F} \mid \epsilon \\ \text{align} & \binom{T}{T} & \to \binom{n}{n} \downarrow \binom{F}{F} \\ \text{deletion} & \binom{T}{Z} & \to \binom{n}{-} \downarrow \binom{F}{F} \\ \text{insertion} & \binom{Z}{T} & \to \binom{-}{n} \downarrow \binom{F}{F} \end{array} \tag{3.7}$$

This grammar explicitly splits $F^{\downarrow}$ and $F^{\rightarrow}$ by applying `iter`. Hence, these parts recurse independently from each other. As shown in Figure 24, aligning two trees can lead to aligning the downward forest of one tree to the forest to the right of the other tree. Given Equation 3.5, the first case corresponds to our first rule in `iter` and the `align` rule. Thus replacing $\binom{T}{T}$ in `iter` we obtain $\binom{F}{F} \to \left(\binom{n}{n} \downarrow \binom{F}{F}\right) \circ \binom{F}{F}$. Here, $\binom{n}{n}$ will give the score for aligning two non-terminals thus $\gamma_{v_2}^{v_1}$. The tree concatenation operators $\downarrow\binom{F}{F}$ and $\circ\binom{F}{F}$ correspond to $F_1^{\downarrow}, F_2^{\downarrow}$ and $F_1^{\rightarrow}, F_2^{\rightarrow}$, respectively. Replacing $\binom{T}{Z}$ and $\binom{Z}{T}$ in the `deletion` and `insertion` rules, we get $\binom{F}{F} \to \left(\binom{n}{-} \downarrow \binom{F}{F}\right) \circ \binom{F}{F}$ and $\binom{F}{F} \to \left(\binom{-}{n} \downarrow \binom{F}{F}\right) \circ \binom{F}{F}$. This corresponds to case 2 and 3 in Equation 3.5. As depicted in Figure 24, given a deleted root in one tree, we recurse by aligning both subforests. In case one of the subforests is empty, it will be replaced by the first tree (the first k trees, respectively) of the remaining forest.

The following shows an example scoring algebra for the case of tree alignment with linear gap costs. No costs are added in the *iter* case but scores obtained from subsolutions are added, given by s_U and s_V. Here, U and V stand for the non-terminal characters whereas u, v are terminals.

$$\begin{aligned} \text{iter}(s_U, s_V) &= s_U + s_V \\ \text{match}(s, \tbinom{u}{v}) &= s + \omega(u, v) \\ \text{deletion}(s, \tbinom{u}{-}) &= s + \omega(u, -) \\ \text{insertion}(s, \tbinom{-}{v}) &= s + \omega(-, v) \\ \text{empty}(\epsilon) &= 0 \\ \text{choice}(s_1, s_2, ..) &= min(s_1, s_2, ..) \\ \text{with } u, v \in \mathcal{A} \text{ and } s, s_U, s_V &\in S \end{aligned} \tag{3.8}$$

Dynamic programming algorithms are based on the order of elements in the input structures which is defined by the index structure in algebraic dynamic programming (ADP). For tree alignment, the index structure corresponds to the preorder in trees defined in Subsection 2.2.1 on page 22.

3.4.1 Multi-tape Tree Alignment

The two-tape version of tree and forest alignment is given above, and it is possible to extend this version to more than two tapes.

Definition 29 (Multiple Tree Alignment)
Let $T_1, T_2, \ldots, T_k$ be ordered trees following Definition 12. The partial order of nodes of each tree sorts parent nodes before their children and children are ordered from left to right. An alignment between trees is an ordered tuple of functions f^i with $i \in [1, .., k]$ which defines a function matching nodes in tree T_i to the consensus tree of the alignment: (i) The set of matched *nodes form a partial order-preserving bijection. That is, if $a = f^i(a), b = f^i(b) \in T_i$ are* matched *with $f^j(a), f^j(b) \in T_j$ the partial orders $a \prec b$ and $f^j(a) \prec f^j(b)$ hold for all nodes a, b and pairs of trees T_i, T_j. (ii) The set of* deleted *nodes form a simple surjection from a node $a \in T_i$ onto a symbol (typically '-') indicating no partner. (iii) For trees T_j, (ii) holds analogously.*

The problem descriptions in Definition 28 and Definition 29 are equivalent. Every forest can be defined as a tree with the roots of the forest as children of a trees' root. Given Definition 28, the mappings π_1 and π_2 can therefore be applied to a forest as well.

Given Definition 29 (i), the partial order is also preserved in case of forests, as the roots of the forests are partially ordered, too. Definition 29 (ii) and (iii) define the order of the deleted nodes that is preserved within the tree. This is also true for forests, and obtained by the mappings π_1 and π_2 defined in Definition 28.

A multiple alignment of forest structures can be constructed given Definition 22 up to Definition 27 and Lemma 1 and Lemma 2. We now show that, given these definitions, the alignment is well-defined and can always be constructed independently of the input structures. We *do not* show here that the resulting alignment is optimal in some sense, as this depends on the optimization function (or algebra) used.

Let $(F_1, \ldots, F_k)$ be an input of k forests. Successive, independent application of the operations specified in Definition 22 up to Definition 27 and Lemma 1 and Lemma 2 to individual input trees yields the multi-tape version of the empty string $(\epsilon_1, \ldots, \epsilon_k)$, i.e., the input is reduced to empty structures. Not every order of application necessarily yields an alignment.

Just as in the case of sequence alignments (Höner zu Siederdissen et al., 2015a) the allowed set of operations needs to be defined for multi-tape input. This amounts in particular to specify which operations act concurrently on multiple tapes. The above-given definitions can be subdivided into two sets. One set of operations takes a forest F and splits off the left- or right-most tree ($F \to F'T$ or $F \to TF'$), where both F' and T are allowed to be empty. In case of T being empty then it holds that $F' = F$. If F' is empty, then this yields a decomposition of F only if F consists of just a single tree. The other set of operations removes atomic elements from a tree and yields a tree or forest, depending on the rule used.

As in Höner zu Siederdissen et al. (2015a), all operations have to operate in "type lockstep" on all inputs simultaneously. That is, either all inputs separate off the left- or right-most tree and yield the trees $(T_1, \ldots, T_k)$ and remaining forests $(F_1, \ldots, F_k)$, or all operations remove a terminal or atomic element, yielding the terminal elements $(n_1, \ldots, n_k)$ and remaining structure $(S_1, \ldots, S_k)$.

Furthermore, an operation $T \to T\,'-'$ is introduced. This operation does not further decompose T but provides a terminal symbol $'-'$ as element n_i. This is analogous to the indel case in string alignments.

Theorem 1 (Alignment)
Every input $(F_1, \ldots, F_k)$ of forests can be deconstructed into $(\epsilon_1, \ldots, \epsilon_k)$ by successive application of the steps outlined above.

Proof 4 *At least one of the decompositions defined above for forests (Definition 22 and Definition 27) or trees (Definition 22 to Definition 25) can always be applied for each input. Due to lockstep handling, all inputs simultaneously separate into a tree and remaining forest (yielding two structures), or apply a rule yielding a terminal element. As such, either all structures are already empty (= ϵ) or at least one structure can be further decomposed yielding a smaller structure.*

The notation of composition of alignments and multi-tape alignments is extended and generalized in Chapter 4.

3.4.2 The Affine Gap Cost Model for Alignments

The simple linear scoring of gaps in alignments as formulated by Needleman and Wunsch (1970) in the original description is often a poor model in computational biology. Instead, one typically uses affine gap cost with a large contribution for opening a gap and small contributions for extending the gaps. The sequence alignment problem with affine gap costs was solved by Gotoh (1982). The corresponding formal grammar, which is based on the notation in Gotoh (1982) and in the version used by Höner zu Siederdissen et al. (2015a), reads

$$\begin{array}{rcl} M & \to & M\binom{u}{v} \mid D\binom{u}{v} \mid I\binom{u}{v} \mid \epsilon \\ D & \to & M\binom{u}{-} \mid D\binom{u}{\cdot} \mid I\binom{u}{-} \\ I & \to & M\binom{-}{v} \mid D\binom{-}{v} \mid I\binom{\cdot}{v} \end{array} \tag{3.9}$$

where u and v are terminal symbols, '$-$' denotes the opening of a gap, and '$.$' denotes the extension of a gap, typically scored differently. Considering only one tape or input dimension, a deletion is denoted by a leading '$-$' followed by a number of '$.$' characters, e.g. a sequence '$-\ldots.$'. The sequence alignment problem with linear gap costs assumes a constant score for each inserted gap symbol '$-$'. This is not a very realistic assumption for biological sequences, as it is more probable to insert or delete a connected substring into genetic material rather than the same amount of singletons being deleted or inserted. Hence, affine gap costs, i.e., a larger penalty for the insertion of the first gap symbol than for the extension of an already existing gap, are commonly used in this field. Consequently, algorithms prefer to insert longer connected gap regions than many singletons. For trees, gap extensions can happen in two directions as a node in a tree has siblings as well as children that are successors given the index structure. Once a node has been aligned to an initial gap symbol ($'-'$) both its siblings and its children are extending the initial gap. Compared to the three rules for matching, deletion and insertion, we now have to deal with seven different cases. In Schirmer (2011) and Schirmer and Giegerich (2011) seven

rules for affine gap costs in forests are formulated based on different modes of scoring: no-gap mode, parent-gap mode and sibling-gap mode. Parent and sibling mode indicate that the preceding node (either parent or sibling node) was considered a deletion. The formal grammar expressing the seven rules for tree alignment with affine costs is shown in Subsection A.1.1 in Appendix A. As the grammar includes a relatively high number of rules, it was further summarized and optimized in two steps with the intermediate version in Subsection A.1.2 and the final version of the grammar is shown in Equation 3.10 and distinguishes between horizontal and vertical gap extension, corresponding to siblings and children in the tree. $\binom{F}{F}$ is the start symbol. Here, we either align the left-most trees or add a gap in one of the roots. Adding a gap in one of the roots leads to a gap extension in the horizontal case, hence $\binom{Q}{Q}$ and in the vertical case, thus $\binom{R}{R}$. The rules on the right-hand side for $\binom{F}{F}$ and $\binom{Q}{Q}$ are the same, as $\binom{Q}{Q}$ is a forest, too. However, $\binom{Q}{Q}$ represents a state where the previous node included a gap symbol. Thus, each gap that is added in the next step will be scored with the gap extension score. For the case $\binom{F}{F}$, no gap was inserted in the direct predecessors, hence every gap added in the next step will be scored with the gap opening score.

$$
\begin{array}{lcl}
\binom{F}{F} & \rightarrow & \binom{T}{T} \circ \binom{F}{F} \mid \binom{T}{Z} \circ \binom{Q}{Q} \mid \binom{Z}{T} \circ \binom{Q}{Q} \mid \epsilon \\
\binom{Q}{Q} & \rightarrow & \binom{T}{T} \circ \binom{F}{F} \mid \binom{T}{Z} \circ \binom{Q}{Q} \mid \binom{Z}{T} \circ \binom{Q}{Q} \mid \epsilon \\
\binom{R}{R} & \rightarrow & \binom{T}{T} \circ \binom{R}{R} \mid \binom{T}{Z} \circ \binom{R}{R} \mid \binom{Z}{T} \circ \binom{R}{R} \mid \epsilon \\
\binom{T}{T} & \rightarrow & \binom{n}{n}\downarrow \binom{F}{F} \\
\binom{T}{Z} & \rightarrow & \binom{n}{-}\downarrow \binom{R}{R} \\
\binom{Z}{T} & \rightarrow & \binom{-}{n}\downarrow \binom{R}{R}
\end{array}
\tag{3.10}
$$

3.4.3 Inside and Outside Grammars

The grammars we defined in previous sections are considered as *inside* grammars in the ADP terminology.

In order to know which cases have to be calculated to obtain probabilities for possible (sub)solutions, an *outside* grammar can be formulated. The outside grammar usually has more rules than the inside grammar and is more complex. In particular, it involves additional non-terminals such as $\binom{F^*}{F^*}$ that can be conceptualized as referring to complements of non-terminals of the inside grammar. Its productions specify how the outside object on the left-hand side can be decomposed. See also Subsection 2.2.5 and Subsection 2.2.7 for more detailed descriptions of inside and outside calculations. Further information on the implementation of inside outside calculations in ADP are given in Appendix A.

In general, the r.h.s. of the productions contain another outside non-terminal as well as inside terminals and non-terminals. The inside objects are the ones that are currently kept fixed whereas the outside objects on the right-hand side define the parts that are used to calculate all possible (sub)solutions with the inside object being fixed. For a detailed discussion of the relationship of inside and outside grammars we refer to Höner zu Siederdissen et al. (2015b) and Durbin et al. (1998).

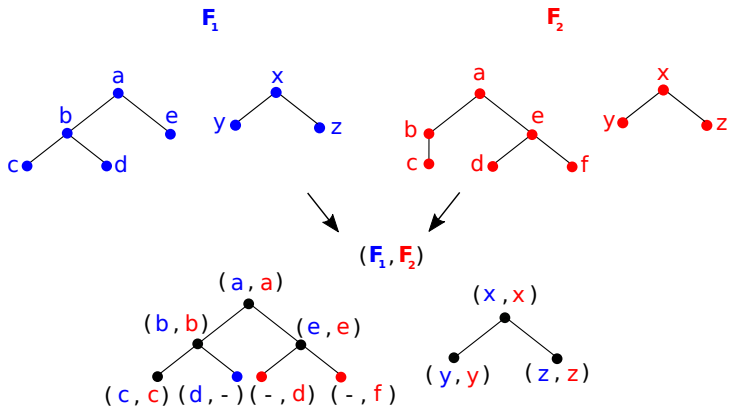


Figure 25: Example of tree alignment on input forests F_1 (blue) and F_2 (red). The aligned structure (F_1, F_2) contains matching nodes (black) and nodes depicting insertions (blue) and deletions (red).

Below, we give the outside grammar in Equation 3.11 (with start symbol $\binom{F^*}{F^*}$), and outside "epsilon" symbols σ – given that σ in an outside grammar terminates with *full* input, not the empty input) for the simple linear-cost tree alignment problem (Equation 3.11) and combine inside and outside grammar to yield match probabilities. Results of inside-outside calculations are probabilities for the alignment of elements of the input structures F_1 and F_2, shown in Figure 26 which correspond to the alignment in Figure 25.

Base cases and calculation of probabilities can be done in different ways. The most basic version is using the inside semiring and probabilities for each production rule instead of usual scores as described in Subsection 2.2.5. However, it is also possible to include partition function and scaling parameter as implement in `ADPfusion` and described in Section A.2.

$$
\begin{aligned}
\tbinom{F^*}{F^*} &\to \tbinom{n}{n} \downarrow \tbinom{T^*}{T^*} \mid \tbinom{n}{-} \downarrow \tbinom{T^*}{Z^*} \mid \tbinom{-}{n} \downarrow \tbinom{Z^*}{T^*} \mid \\
\tbinom{F^*}{F^*} &\to \tbinom{T}{T} \circ \tbinom{F^*}{F^*} \mid \tbinom{T}{Z} \circ \tbinom{F^*}{F^*} \mid \tbinom{Z}{T} \circ \tbinom{F^*}{F^*} \mid \sigma \\
\tbinom{T^*}{T^*} &\to \tbinom{F^*}{F^*} \circ \tbinom{F}{F} \\
\tbinom{T^*}{Z^*} &\to \tbinom{F^*}{F^*} \circ \tbinom{F}{F} \\
\tbinom{Z^*}{T^*} &\to \tbinom{F^*}{F^*} \circ \tbinom{F}{F}
\end{aligned}
\tag{3.11}
$$

The combined inside-outside algorithm with an affine gap cost model can be implemented in complete analogy to the linear model. One usually designs the inside grammar based on the recursion equations and the outside grammar is constructed automatically. This yields an algorithm that computes the match probabilities using the affine gap cost model. Further examples are shown in Section A.2.

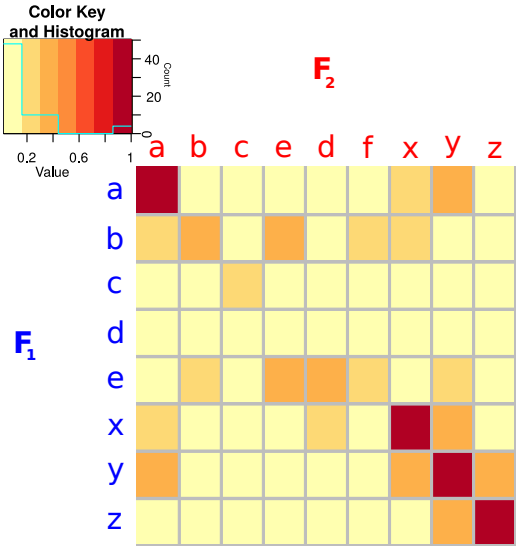


Figure 26: Example of Inside-Outside calculation for the tree alignment shown in Figure 25 on forests F_1 (red) and F_2 (blue). Cells in the heatmap show probabilities for the alignment of corresponding nodes whereas dark colors show higher probabilities.

3.5 Tree Editing

The string-to-string correction problem can be generalized to forests. To this end, edit operations need to be explained for trees. We consider substitution (or relabeling), insertion, and deletion and associate them with costs γ_y^x, $\gamma_z^\varnothing$, or $\gamma_\varnothing^z$. Figure 27 gives an example for tree editing of one tree into another tree. Here, one can see that nodes or positions in the tree can be edited several times, e.g. the position of node b (blue) which is deleted such that its child, node c, is moved on its position. Later, node g (red) is inserted between the root and node c.

In this section, we explain our formal grammar for the string editing algorithm on trees instead of strings. Comparing and editing strings is a well-known algorithm in computer science applications, and hence, we expanded this application to trees as inputs described by a formal grammar.

The grammar is based on Zhang and Shasha (1989) which shows one possible solution for the tree editing dynamic programming (DP) algorithm. There exist other ways which try to minimize the search space of possible solutions by explicitly choosing where to start the next iteration step (Schwarz et al., 2017; Pawlik and Augsten, 2016; Bringmann et al., 2018).

Definition 30 (Tree Editing, *cf.* Bille (2005))
A mapping between two ordered forests F_1 to F_2 is a binary relation $M \in V(F_1) \times V(F_2)$ between the vertex sets of the two forests such that for pairs $(x, y), (x', y') \in M$ holds

i) $x = x'$ if and only if $y = y'$. (one-to-one condition)

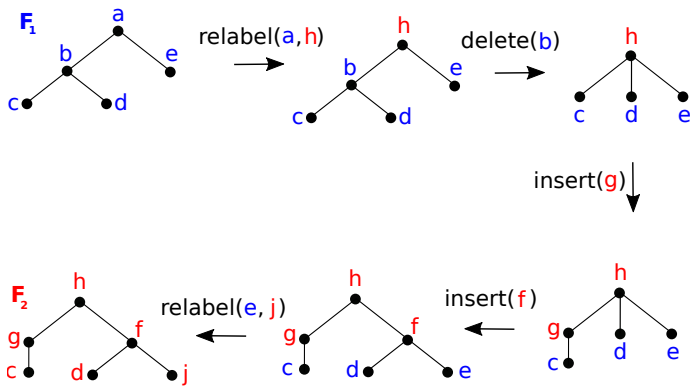


Figure 27: Example of the tree editing algorithm, transforming tree F_1 into tree F_2. Operations are relabel, delete and insert.

ii) x is an ancestor of x' if and only if y is an ancestor of y'. (ancestor condition)

iii) x is to the left of x' if and only if y is to the left of y'. (sibling condition)

The one-to-one condition implies that for each $x \in F_1$ there is a unique "partner" $y \in F_2$, i.e., $(x, y) \in M$, or x has no matching partner at all. With each mapping we can associate the cost

$$\gamma(M) = \sum_{(x,y)\in M} \gamma_y^x + \sum_{y:(x,y)\notin M} \gamma_y^{\varnothing} + \sum_{x:(x,y)\notin M} \gamma_{\varnothing}^x \tag{3.12}$$

Individual edit operations correspond to "elementary maps". Maps can be composed in a natural manner. Thus every edit script corresponds to a map. Conversely every map can be composed of elementary maps, and thus corresponds to an edit script. Furthermore, the cost of maps is subadditive under composition. As a consequence, minimum cost mappings are equivalent to the minimum cost edit scripts (Tai, 1979).

The problem of minimizing $\gamma(M)$ has a dynamic programming solution. For a given forest F and a root node v in F, we denote by $F - v$ the forest obtained by deleting v and $F \setminus T(v)$ is the forest obtained from F by deleting with v all descendants of v. Note that $T(v) - v$ is the forest consisting of all trees whose roots are the children of v. Equation 3.13 shows the recursion equations.

$$D(F_1, F_2) = \min \begin{cases} D(F_1 - v_1, F_2) + \gamma_{\varnothing}^{v_1} \\ D(F_1, F_2 - v_2) + \gamma_{v_2}^{\varnothing} \\ D(T(v_1) - v_1, T(v_2) - v_2) + \gamma_{v_2}^{v_1} \\ \qquad + D(F_1 - T(v_1), F_2 - T(v_2)) \end{cases} \tag{3.13}$$

with $D(\emptyset, \emptyset) = 0$ for two empty forests. A key issue is to implement this algorithm in such a way that only certain classes of subforests need to be evaluated. Here, forests are

decomposed into the right-most tree and the remaining forest, which is not specified in the recursion equations in Equation 3.13. The corresponding tree editing grammar $\mathcal{E}$ reads

$$\begin{array}{lcl} \binom{T}{T} & \to & \binom{F}{F} \downarrow \binom{n}{n} \\ \binom{F}{F} & \to & \binom{F}{F} \circ \binom{T}{T} \;\Big|\; \binom{F}{F} \downarrow \binom{x}{-} \;\Big|\; \binom{F}{F} \downarrow \binom{-}{x} \;\Big|\; \epsilon \end{array} \tag{3.14}$$

Note that the empty symbol '$-$' acts as neutral element for the concatenation operators, which we take to act component-wise. The grammar is based on the tree editing algorithm of Zhang and Shasha (1989), for which several more efficient implementations exist. In particular, Dulucq and Tichit (2003) provides a detailed analysis of the Zhang-Shasha algorithm.

Given the formal grammar in Equation 3.14 describing the tree editing problem, we modeled the individual cases of the recursion in Equation 3.13. Replacing $\binom{T}{T}$ in the iteration rule we get $\binom{F}{F} \to \binom{F}{F} \circ (\binom{F}{F} \downarrow \binom{n}{n})$. This matches the third case of the recursion equation, as we add the score for a match and recurse on the forests of the children and the remaining forests. The other two cases of the recursion equation delete the right-most root from one of the current forests using Definition 26, add the score for a deletion or insertion and recurse on the remaining forests.

Tree editing and tree alignment make use of the same set of decomposition operators but differ in their formal grammars and algebras. Many related algorithms on trees, such as the ones described in Kan et al. (2014) and Kuboyama (2007), can also be expressed by formal grammars with these decomposition operators. Depending on the details of the problem at hand, the grammar might become more complicated; nevertheless, they usually show striking similarities to the corresponding recursion equations. The underlying index structure for tree editing corresponds to the postorder of trees defined in Subsection 2.2.1.

3.5.1 Outside Grammar

Analogously to the outside grammar for tree alignment (Equation 3.11), we give the outside grammar for tree editing with start symbol $\binom{F^*}{F^*}$ and outside "epsilon" symbols σ – given that σ in an outside grammar terminates with *full* input, not the empty input.

$$\begin{array}{l} \binom{F^*}{F^*} \to \binom{n}{n} \downarrow \binom{T^*}{T^*} \;\Big|\; \binom{F^*}{F^*} \downarrow \binom{n}{-} \;\Big|\; \binom{F^*}{F^*} \downarrow \binom{-}{n} \;\Big|\; \binom{F^*}{F^*} \circ \binom{T}{T} \;\Big|\; \sigma \\ \binom{T^*}{T^*} \to \binom{F^*}{F^*} \circ \binom{F}{F} \end{array} \tag{3.15}$$

3.5.2 Affine Gap Costs

The following grammar (Equation 3.16) describes tree editing with affine gap costs. Here, non-terminals R and Q describe the gap modes.

$$\begin{array}{lcl} \binom{F}{F} & \to & \binom{F}{F} \circ \binom{T}{T} \;\Big|\; \binom{R}{R} \circ \binom{x}{-} \;\Big|\; \binom{Q}{Q} \circ \binom{-}{x} \;\Big|\; \epsilon \\ \binom{Q}{Q} & \to & \binom{F}{F} \circ \binom{T}{T} \;\Big|\; \binom{Q}{Q} \circ \binom{-}{x} \;\Big|\; \epsilon \\ \binom{R}{R} & \to & \binom{F}{F} \circ \binom{T}{T} \;\Big|\; \binom{R}{R} \circ \binom{x}{-} \;\Big|\; \epsilon \\ \binom{T}{T} & \to & \binom{n}{n} \downarrow \binom{F}{F} \end{array} \tag{3.16}$$

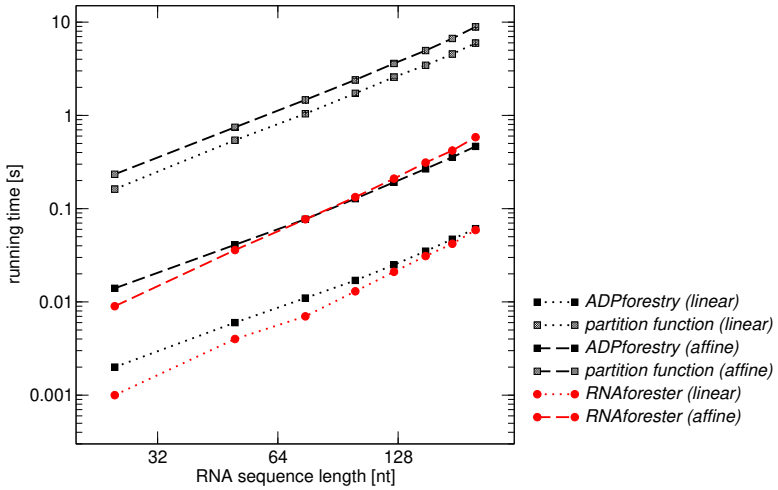


Figure 28: Comparison of the running times (in seconds) of `RNAforester` and `ADPforestry` measured as an average over 50 instances of random RNA secondary structures of different lengths using linear and affine gap costs, respectively. The benchmarks were performed on an Intel(R) Core(TM) i5-4570 machine with 32 Gb memory. `RNAforester` does not implement an inside-outside (partition function) version of tree alignment.

3.6 Benchmarking against `RNAforester`

`RNAforester` (Höchsmann, 2005), which is designed to compare RNA secondary structures, is the most widely used implementation of tree alignment. In order to compare our Haskell implementation `ADPforestry` with `RNAforester` we therefore use RNA secondary structures computed with `RNAfold` (Lorenz et al., 2011a) for 50 pairs of RNA sequences of different lengths. The resulting pairs of secondary structures in "dot-bracket" notation together with the RNA sequences directly serve as input for `RNAforester`. We measured the performance of both programs with both for linear and affine gap costs models. In addition, we show running times for the inside-outside (partition function) algorithms implemented in `ADPforestry`.

Benchmarking results are compiled in Figure 28. `ADPforestry` at present only implements a simple scoring scheme and none of the elaborate RNA specific scoring schemes provided by `RNAforester`. It also does not attempt to reproduce the detailed, largely RNA-specific output of `RNAforester`. In contrast to `ADPforestry`, `RNAforester` only solved the optimization version of the tree alignment problem. An alternative implementation of the inside-outside algorithms to compute the a posteriori probabilities of all possible alignment edges is not available at all.

The comparably large effort for the calculation of probabilities is explained by (a) the computation of not only the inside recursion as in case of optimization but also of the corresponding, automatically generated outside recursion, and (b) the fact that the outside recursion is more complex than the inside recursion. Here the inside grammar for tree alignment with linear costs contains 6 rules whereas the corresponding outside

grammar is composed out of 8 rules. For the affine case, the inside grammar has 16 rules, the outside grammar 25.

3.7 Software Availability

Implementations for the algorithms discussed here are available on `hackage` and `github`. The `ADPfusionForest` framework extensions are available at `http://hackage.haskell.org/package/ADPfusionSet` and prototypical implementations of the forest-based algorithms at `http://hackage.haskell.org/package/Forestry`. The latter are accompanied by a simple `RNAforester` variant.

3.8 Conclusion

Tree comparison has many applications and there exist several approaches in the literature that optimize existing algorithms regarding the size of the search space or time and memory requirements. Based on existing optimizations, we formalized the algorithms tree editing (Zhang and Shasha, 1989) and tree alignment (Schirmer, 2011; Schirmer and Giegerich, 2011) such that they can be written as formal grammars as it has been done for string comparison (Höner zu Siederdissen et al., 2015a). Compared to strings, trees can be traversed in two directions, thus our data structure is 2-dimensional. Each grammar consists of terminals and non-terminals, whereas the terminals are single nodes and the non-terminals specify a tree and a forest.

Giegerich and Touzet (2014) developed a system of inverse coupled rewrite systems (ICOREs) which unify dynamic programming on sequences and trees. Rules written in this system bear a certain resemblance to our system. We are not aware of an implementation of ICORES, however, making comparisons beyond the formalism is difficult. All of these approaches use trees and/or forests as input structures. Aims and methods, however, are different.

In addition to the linear gap cost version of the algorithms, we developed grammars that provide variants with affine gap costs. Compared to simple, linear cost functions, additional non-terminals are required to distinguish between the initialization of a gap, or gap opening, and gap extension.

For each (inside) grammar, the corresponding outside grammar of the original algorithm (Höner zu Siederdissen et al., 2015b) can be automatically calculated. Using inside and outside versions, we can specify match probabilities for each pair of subtrees. The inside and outside grammar together thus allow the calculation of ensemble properties such as the probability of each pair of local tree roots to be paired with each other.

Combining ADP with tree and forest structures as input, we are now able to apply single-tape and multi-tape DP algorithms on tree-like data. Together with the inclusion of Inside-Outside algorithms, the search space of DP algorithms on trees and forests can be explored broadly. As grammar and algebra can be changed easily, it is possible to compare results based on distinct grammars or cost functions.

CHAPTER 4

Dynamic Programming with Alignments on General Data Structures

Contents

Despite the immense practical importance of alignments (see also Subsection 2.3.1), they have received very little attention as mathematical structures in the past. The most comprehensive treatment is the Technical Report (Morgenstern et al., 1999) which considers (pairwise) alignments as binary relations between sequence positions that represent matchings and preserve order. The aim here is not to construct concrete alignment algorithms but the systematic generalization of alignments from strings to more general discrete objects. Such a generalization still supports the recursive construction that underlies the exact dynamic programming algorithms employed to compute score-optimal alignments in the totally ordered case. Following earlier work (W. Otto et al., 2011), the language used here is closer to graph theory than the presentation of Morgenstern et al. (1996) and Morgenstern et al. (1999).

This chapter is based on Berkemer et al. (2019) titled *Compositional Properties of Alignments* and will almost completely disregard the scoring of alignments and instead focus on the structure of (multiple) alignments as combinatorial objects. In the following sections, we explore the consequence of relaxing some of the axioms to cover partial orders in general. Additionally, this chapter is mainly concerned to ensure that alignments of alignments are well-defined as a foundation for progressive alignment procedures, and that decompositions into blocks exist that can form the basis of divide-and-conquer approaches to aligning partially ordered sets. Following a brief discussion of the view of alignments as compositions of pairwise matching relations, the chapter continues by further generalizing the formalism to include first ordered trees, then directed and undirected graphs, and finally essentially arbitrary finite spaces that admit well-behaved subspace constructions.

4.1 Formal Definitions of Sequence Alignments

Alignments are usually constructed from strings or other totally ordered inputs (see also Subsection 2.1.1 and Subsection 2.2.1), hence the columns of the resulting alignment are usually also treated as a totally ordered set. Consecutive insertions and deletions, however, are not naturally ordered relative to each other:

```
gugugu--acgggcca        guguguac--gggcca
gucuguug--gggccc        gucugu--uggggccc
```
(4.1)

are alignments that are equivalent under most plausible scoring models. The idea to consider alignment columns as partial orders was explored systematically in C. Lee et al. (2002) and a series of follow-up publications (C. Lee, 2003; Grasso and C. Lee, 2004). Here, (mis)matches are considered as an ordered backbone, with no direct ordering constraints between an insertion and a deletion. The resulting alignments are then represented as directed acyclic graphs (DAGs) (see also Subsection 2.2.1), more precisely, as the Hasse diagrams of the partial order. The key idea behind the `POA` software (C. Lee et al., 2002) is that a sequence of DAGs can be used as an input to a modified version of the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970). Recently this idea has been generalized to the problem of aligning a sequence to a general directed graph (Rautiainen and Marschall, 2017; Vaddadi et al., 2017).

Suppose we are given a set S of $|S| \geq 1$ sequences with not necessarily equal length. For $s \in S$ we write s_i for the i-th position in s, and $|s|$ denotes the length of s, i.e., the number of positions. The most common representation of an alignment is a rectangular

matrix whose rows are indexed by the sequences and whose columns are indexed by integers $i \in [1, L]$, where L is the number of alignment columns. Each sequence is then associated with a strictly monotonically increasing function $\alpha_s : [1, |s|] \to [1, L]$ such that for each $i \in [1, |s|]$, $\alpha_s(i)$ is the index of the column containing s_i. The alignment matrix contains a gap symbol in row s and column k whenever $\alpha_s^{-1}(k) = \emptyset$, otherwise, the matrix element is $s_{\alpha_s^{-1}(k)}$. The following example has already been shown in Subsection 2.1.2 on page 17, however, for reasons of consistency and completeness, it is shown again here:

```
a   001111100
b   0011011--
c   --1000100
i   123456789
```
(4.2)

The rows correspond to sequences a, b and c and row i indicates the indices of alignment columns. We have $\alpha_A(i) = i$ for $1 \leq i \leq 9$, $\alpha_B(i) = i$ for $1 \leq i \leq 7$, and $\alpha_C(i) = i + 2$ for $1 \leq i \leq 7$. For the 8th column of the example we have $\alpha_a^{-1}(8) = 8$, $\alpha_b^{-1}(8) = \emptyset$, $\alpha_c^{-1}(8) = 6$; hence the entries in the 8th column are $a_{\alpha_a^{-1}(8)} = a_8 = \mathtt{0}$, $-$ because $\alpha_b^{-1}(8) = \emptyset$, and $c_{\alpha_c^{-1}(8)} = c_6 = \mathtt{0}$. It will be convenient in the discussion below to also consider single sequences as (trivial) alignments, using the identity on $[1, |s|]$.

The actual values of the sequence elements, i.e., the s_i are of course important to determine the scoring. For our purposes, however, they are irrelevant, since we will only be interested in the structure of the alignments. It therefore suffices to consider the *sequence positions* $X_s := [1, |s|]$ for each input sequence and their arrangement in the alignment columns. This information is completely contained in the functions α_s. We can therefore "forget" about almost all the details about the sequence s except its length, which by construction satisfies $|X_s| = |s|$. From here on, we can therefore treat s simply as an index used solely to enumerate the elements of S. We will use the symbol $\bullet$ to indicate that a particular cell in the alignment matrix is occupied, while $-$ indicates gaps. The $\bullet$ eventually will become vertices in a graph representation.

For our purposes the set of sequence positions X_s is simply a finite ordered set. To emphasize this fact, and to make generalizations below more transparent, we write $(X_s, <_s)$ to explicitly expose the order relation on X_s. For a given set of sequences, furthermore, we will need the set of all sequence positions defined as the disjoint union $X := \bigcup\!\!\!\cdot_{s \in S} X_s$ of all sequence positions. The structure of an alignment with L columns is completely determined by the function $\omega : [1, L] \to \prod_{s \in S}(X_s \cup \{-\})$ such that $\omega(k) = (\omega_s(k) | s \in S)$, where

$$\omega_s(k) = \begin{cases} -, & \text{if } \alpha_s^{-1}(k) = \emptyset \\ \alpha_s^{-1}(k) & \text{if } \alpha_s^{-1}(k) \neq \emptyset \end{cases}$$

Thus ω plays the role of a (slightly modified) inverse of α. It is customary, furthermore, to exclude alignment columns that consist entirely of gap symbols:

Definition 31 (Proper Alignment)
An alignment on $X = \bigcup\!\!\!\cdot_{s \in S} X_s$ defined by ω is proper *if there is no k such that $\omega_s(k) = -$ for all $s \in S$.*

Given ω, we construct a graph with vertex set $X = \bigcup\!\!\!\cdot_{s \in S} X_s$ and edge set A such that $xy \in A$ if there is $k \in [1, L]$ and distinct sequences s and t such that $\omega_s(k) = x$ and

$\omega_t(k) = y$. In other words, positions x and y are joined by an edge if and only if x and y appear in the same column of the alignment. We call this graph the *alignment graph*.

Lemma 4 *Consider an alignment on $X = \bigcup\!\!\!\cdot_{s\in S} X_s$ determined by ω. Suppose $x \in X_r$, $y \in X_s$, and $z \in X_t$ and both xy and xz are edges in the alignment graph. Then*

(i) r, s, and t are pairwise distinct.

(ii) yz is also an edge in the alignment graph.

Proof 5 *Property (i) follows immediately from the requirement that α_s is strictly monotonically increasing, i.e., any two positions of the same sequence are mapped to distinct alignment columns. Property (ii) follows directly from the definition. If xy and xz are edges, then x, y, and z are located in the same alignment column and thus yz is an edge of the alignment graph.*

The alignment graph therefore is the disjoint union of complete graphs such that every connected component (which is a clique) contains at most one element of each of the input sequences X_s. Every clique thus corresponds to an alignment column. We write $\mathcal{C}(X, A)$ for the set of alignment columns, which for convenience we identify with their vertex sets. More precisely, $Q \in \mathcal{C}(X, A)$ is an alignment column if and only if there are $x \in Q$, $k \in [1, L]$, and $s \in S$ such that $x = \omega_s(k) \neq -$. In particular, for each $s \in S$ we have either $Q \cap X_s = \emptyset$ or $Q \cap X_s = \{\omega_s(k)\}$ for some k.

The alignment graph is consistent with the input orders $<_s$ on X_s, $s \in S$ in the following sense:

Lemma 5 *Let Q' and Q'' be two distinct connected components of the alignment graph with vertex set X determined by ω and suppose there are $s, t \in S$ such that $x_s \in Q' \cap X_s$, $y_s \in Q'' \cap X_s$, $x_t \in Q' \cap X_t$, and $y_t \in Q'' \cap X_t$. Then $x_s <_s y_s$ if and only if $x_t <_t y_t$.*

Proof 6 *By Lemma 4, two vertices x_s and x_t are in the same connected component Q if and only if they are in the same column, i.e., if $\alpha(x_s) = \alpha(x_t) =: k'$. Analogously, $\alpha(y_s) = \alpha(y_t) =: k''$. By monotonicity of α_s and α_t, we therefore have $x_s <_s y_s$ if and only if $k' < k''$, which in turn is true if and only if $x_t <_t y_t$.*

In particular, we may conclude:

Observation 1 *Consider an alignment on X determined by ω. Then there exists an order on the alignment columns such that $Q' <_A Q''$ implies $x <_s y$ whenever $x \in Q' \cap X_s$ and $y \in Q'' \cap X_s$.*

Proof 7 *By construction, the alignment columns are ordered. Lemma 5 implies that this order is consistent with the order $<_s$ of each X_s.*

In the following it will be convenient to write each element of X as a pair that explicitly specifies the input sequence from which it derives. That is, we write $(a, i) \in X$ for $i \in X_a$ and $a \in S$.

The simple observations of this section suggest to *define* an alignment by means of an alignment graph with a suitable order of the columns. The following definition rephrases the approach taken e.g. in Stoye et al. (1997), Morgenstern et al. (1996), and Morgenstern et al. (1999) in a form that will be most convenient for further generalizations:

Definition 32 (Total Alignment, W. Otto et al. (2011))
A total alignment of a finite collection of finite totally ordered sets $(X_s, <_s)$, $s \in S$, is a triple $(X, A, <)$ where $X := \bigcup_{s\in S} X_s$, (X, A) is an undirected, loop-free graph with vertex set X with $\mathcal{C}(X, A)$ being the set of its connected components, and $<$ is a total order relation on $\mathcal{C}(X, A)$ such that the following conditions are satisfied.[1]

(1) *$Q \in \mathcal{C}(X, A)$ is a complete subgraph of (X, A).*

(2) *If $(a, i) \in Q$ and $(a, j) \in Q$ then $i = j$.*

(4) *If $(a, i), (b, j) \in P$ and $(a, k), (b, l) \in Q$ with $i <_a k$ then $j <_b l$.*

(5) *If $(a, i) \in P$, $(a, j) \in Q$ and $(a, i) <_a (a, j)$ then $P < Q$.*

As above, the connected components of the alignment graph (X, A) play the role of the alignment columns. Condition (2) ensures that every alignment column contains at most one element of each ordered set X_a. Conversely, every element (a, i) is contained in exactly one connected component, i.e., alignment column. Condition (4) requires that alignment columns do not cross. Condition (5) ensures that the restriction of order on the columns to each row recovers the order $(X_a, <_a)$. A bit more formally, this can be phrased as follows:

Observation 2 *Let $(X, A, <)$ be an alignment, $P, Q \in \mathcal{C}(X, A)$, $P \cap X_a = \{(a, i)\}$, and $Q \cap X_a = \{(a, j)\}$. Then $P < Q$ if and only if $(a, i) <_a (a, j)$.*

A well known observation in the theory of alignments is that Conditions (4) and (5) in general only specify a partial order but not a total order of the alignment columns:

Lemma 6 *Let (X, A) be an alignment graph and denote by $\prec$ the relation defined for all $P, Q \in \mathcal{C}(X, A)$ by $P \dot{\prec} Q$ whenever there is an $a \in S$ such that $(a, i) \in P$, $(a, j) \in Q$ and $i < j$. Then the transitive closure $\prec$ of $\dot{\prec}$ is a partial order on $\mathcal{C}(X, A)$.*

Proof 8 *By construction, $\dot{\prec}$ is antisymmetric. By definition $P \prec Q$ if and only if there is a sequence of columns $P = Q_0 \dot{\prec} Q_1 \dot{\prec} \ldots \dot{\prec} Q_k = Q$. Since the sequence of elements (a, i) belonging to the same X_a is strictly increasing with the column index j for each a along any such sequence of columns, it follows that the transitive closure of $\prec$ is still antisymmetric. Thus $\prec$ is a partial order.*

As an immediate consequence, there is also a (not necessarily unique) total order $<$ of the alignment columns, obtained as an arbitrary linear extension of $\prec$, which by construction satisfies

$$P < Q,\ (a, i) \in P, \text{ and } (a, j) \in Q \quad \text{implies} \quad i < j. \tag{4.3}$$

We summarize this reasoning in

[1] There is no condition (3) due to synchronization with the definitions for partial orders defined later.

Theorem 2 *Let (X, A) be an alignment graph for $X = \bigcup_{s \in S} X_s$ and conditions (1), (2), and (4) of Definition 32 are satisfied. Then there exists a total order $<$ on $\mathcal{C}(X, A)$ satisfying condition (5), i.e., such that $(X, A, <)$ is a total alignment.*

Theorem 2 provides the justification for considering alignment graphs with an order on the columns instead of the matrix representation defined by ω. Obviously $(X, A, <)$, or more precisely the order $<$ of the alignment columns completely defines α, ω, and L *provided* we require that there are no alignment columns consisting entirely of gap symbols.

Before we proceed, a few remarks are in order: In this setting the actual data associated with the sequence element (a, i), whether it is simply the i-th letter of input sequence a or an extensive entry at position i of the list a, is treated as a label that influences only the scoring but not the structure of the alignment. This separation between the underlying (index) structure and the data associated with them is also used in algebraic dynamic programming approaches to alignments (Höner zu Siederdissen et al., 2015b; Berkemer et al., 2017b), where the structure of the recursions depends only on the possible alignments $(X, A, <)$ for a given set X, while the scoring depends on the labeling of X. In order to treat (partially) local alignments it is necessary to distinguish aligned and "unaligned" columns. Each unaligned column may contain only a single element, i.e., every unaligned position is considered as an insertion relative to all other elements of X. Whether a position is aligned or unaligned affects only the scoring, hence at the level of alignment graphs we do not need to concern ourselves with a distinction of local, partially local, and global alignments.

4.2 Alignments of Partially Ordered Sets

Since the alignment of totally ordered sets in general only specifies a partial order of columns but not a total order, it seems natural to ask whether the concept of alignments and alignment graphs can be extended to partial orders instead of total orders on inputs. From here, one therefore considers a collection of finite partial orders $(X_a, \prec_a)$, $a \in S$, $|S| \geq 1$. As a generalization of Definition 32 we consider

Definition 33 (PO Alignment)
A partial order (PO) alignment of X is a triple $(X, A, \prec)$ where (X, A) is a graph and $\prec$ is a partial order on the set of connected components $\mathcal{C}(X, A)$ such that

(A1) *$Q \in \mathcal{C}(X, A)$ is a complete subgraph of (X, A).*

(A2) *If $(a, i) \in Q$ and $(a, j) \in Q$, then $i = j$.*

(A3) *If $(a, i) \in P$, $(a, j) \in Q$ for some $P, Q \in \mathcal{C}(X, A)$ and $(a, i) \prec_a (a, j)$ then $P \prec Q$.*

(A4) *$P \prec Q$, $(a, i) \in P$ and $(a, j) \in Q$ implies $(a, i) \prec_a (a, j)$ or (a, i) and (a, j) are incomparable w.r.t. $\prec_a$.*

Condition (A3) constrains the partial order on the columns to respect the partial order of the rows. Condition (A4) insists that columns also must not cross indirectly.

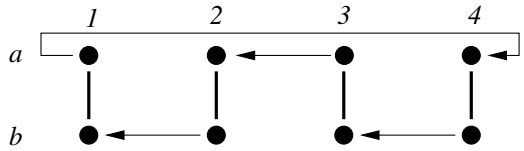


Figure 29: Property (A4*) is not sufficient to ensure the existence of a partial order $\prec$ on $\mathcal{C}(X, A)$. Consider the partial orders $(a, 4) \prec_a (a, 1)$ and $(a, 2) \prec_a (a, 3)$ and $(b, 1) \prec_b (b, 2)$ and $(b, 3) \prec_a (b, 4)$, with alignment columns $\{(a, i), (b, i)\}$ for $i = 1, 2, 3, 4$. Clearly (A2), (A3), and (A4*) holds, but the directed cycle shows that no partial order on the columns exists that is consistent with both partial orders.

If all $(X_a, \prec_a)$ are totally ordered then condition (A4) implies the non-crossing condition (4) because (b, j) and (b, l) cannot be incomparable w.r.t. $\prec_b$, and thus the required partial order $\prec$ is obtained as the transitive closure of the relative order of any two columns. Definitions 32 and 33 therefore coincide for totally ordered rows.

Condition (A4) obviously implies the following generalization of (4):

(A4*) $(a, i), (b, j) \in P$ and $(a, k), (b, l) \in Q$ and $(a, i) \prec_a (a, k)$ implies $(b, j) \prec_b (b, l)$ or (b, j) and (b, l) are incomparable w.r.t. $\prec_b$ $\forall P, Q \in \mathcal{C}(X, A)$.

However, (A4*) is not sufficient to guarantee that the alignment columns form a partially ordered set. A counterexample is shown in Figure 29. It is therefore necessary to require the existence of the partial order $\prec$ on the alignment columns $\mathcal{C}(X, A)$ as an extra condition in Definition 33.

The existence of (non-trivial) alignments of any collection of finite partial orders $(X_s, \prec_s)$, $s \in S$, is easy to see: each of the partial orders can be linearly extended to a total order $(X_s, <_s)$. Any alignment of these total orders is also an alignment of the underlying partial orders, with a suitable partial order of the columns given by Lemma 6.

Before we proceed we briefly remark that at the level of our discussion we do not need to concern ourselves with the distinction of global and local alignments. In order to model a partially local alignment of posets we consider the set $\mathcal{A}$ of aligned columns and a partition of the set of "unaligned columns" into two not necessarily non-empty subsets $\mathcal{P}$ and $\mathcal{S}$ such that for all $U \in \mathcal{P}$, $V \in \mathcal{A}$ and $W \in \mathcal{S}$ it holds that $W \not\preceq V$ and $V \not\preceq U$, i.e., no "unaligned" suffix column precedes an aligned column, and no "unaligned" prefix column succeeds an aligned column. "Unaligned" prefix columns belonging to different rows $(X_a, \prec_a)$ are considered mutually incomparable; the same is assumed for "unaligned" suffix columns. With the caveat that "unaligned" columns need to be marked as such, there is again no structural difference between local and global alignments.

The projection of $(X, A, \prec)$ onto a row $a \in S$ is obtained as the set $\pi_a(X) := \{(a, i) \in X_a | \exists Q \in \mathcal{C}(X, A) : (a, i) \in Q\}$ endowed with the partial order $\prec_a^\pi$ such that $(a, i) \prec_a^\pi (a, j)$ whenever there are columns $P, Q \in \mathcal{C}(X, A)$ with $P \prec Q$. A potential shortcoming of Definition 33 is that it does not guarantee that $(\pi_a(X), \prec_a^\pi) = (X_a, \prec_a)$. It is therefore of interest to consider a (much) stronger version of axiom (A4):

(A5) $P \prec Q$, $(a, i) \in P$ and $(a, j) \in Q$ implies $(a, i) \prec_a (a, j)$; $\forall P, Q \in \mathcal{C}(X, A)$.

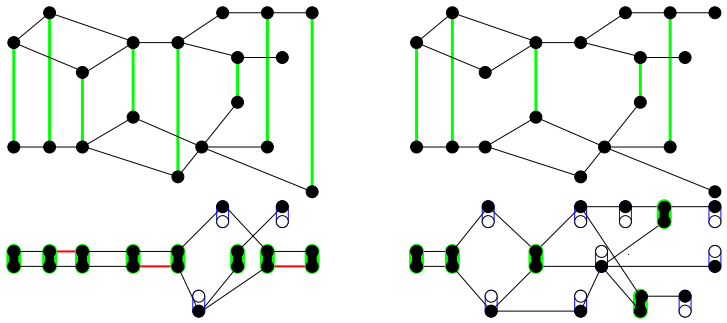


Figure 30: Top: Pairwise alignments of partially ordered sets. Thin black edges show the Hasse diagram, to be read from left to right. Alignment edges are shown in green.
Bottom: The induced partial order of the alignment columns with corresponding points vertically aligned. The partial order is again shown as a Hasse diagram, with superfluous edges omitted. Both the l.h.s. and the r.h.s. example satisfy (A4), i.e., none of the order relations $\prec_1$ and $\prec_2$ is violated in the alignment. The red edges highlight two comparabilities introduced by partial order of the columns that are absent in the input posets. Red edges therefore imply a violation of condition (A5). Hence the l.h.s. alignment violates (A5), while the r.h.s. alignment does not.

First we note that (A5) implies (A4). The definition of the projection of $(X, A, \prec)$ to a row $a \in S$ then immediately implies

Observation 3 *Suppose a* $(X, A, \prec)$ *satisfies* (A1)*,* (A2)*. Then* (A5) *is equivalent to* $(\pi_a(X), \prec_a^\pi) = (X_a, \prec_a)$*.*

Observation 3 furthermore implies that (A4) and (A5) are equivalent if all $(X_a, \prec_a)$ are totally ordered. In general this is not the case, however, as the example in Figure 30 shows.

The following simple, technical result is a generalization of Lemma 6, showing that condition (A5) is sufficient to guarantee the existence of a partial order on the columns.

Lemma 7 *Let* (X, A) *be a graph with connected components* $\mathcal{C}(X, A)$ *satisfying* (A1) *and* (A2)*. Let* $\dot\prec$ *denote the transitive closure of the relation* $\prec$ *defined by* (A3)*, i.e.,* $P \dot\prec Q$ *whenever* $(a, i) \in P$*,* $(a, j) \in Q$ *and* $(a, i) \prec_a (a, j)$ *then* $P \prec Q$*;* $\forall P, Q \in \mathcal{C}(X, A)$*. Finally assume that axiom* (A5) *holds. Then* $\prec$ *is a partial order on* $\mathcal{C}(X, A)$*.*

Proof 9 *It suffices to show that* $\dot\prec$ *is antisymmetric. It is clear from the construction that by* (A5) *we know that* $\prec$ *is antisymmetric. If* $\dot\prec$ *is not antisymmetric, then there is a finite sequence of columns* P_i*,* $i = 0, \ldots, k$ *such that* $P_0 \dot\prec P_1 \dot\prec \ldots \dot\prec P_k \dot\prec P_0$ *and any two consecutive columns* P_i *and* P_{i+1} *have a pair of entries, say* $(a_i, h) \in P_i$ *and* $(a_i, h') \in P_{i+1}$*, in the same row. For the transitive closure this would imply both* $(a_i, h) \prec (a_i, h')$ *from* $(a_i, h) \dot\prec (a_i, h')$ *and* $(a_i, h') \prec (a_i, h)$ *by going around the cycle, contradicting axiom* (A5)*.*

Finite partial orders $(X_a, \prec_a)$ are equivalent to finite directed transitive acyclic graphs. The projection property of Observation 3, can be expressed in graph-theoretical terms in the following manner:

Observation 4 *Let $(X, A, \prec)$ be an alignment of partial orders $(X_a, \prec_a)$, $S' \subseteq S$ a subset of columns, and $\mathcal{Q}' \subseteq \mathcal{C}(X, A)$ such that $X_a \cap Q \neq \emptyset$ for all $a \in S'$ and $Q \in \mathcal{Q}'$. Then the graph with vertex set $\mathcal{Q}'$ and directed edges whenever $P \prec Q$ is an induced subgraph of (the graph representation of) $(X_a, \prec_a)$.*

Thus the set of alignment columns $\mathcal{Q}'$ defines an *induced common subgraph* of the transitive acyclic graphs $(X_a, \prec_a)$ in $a \in S'$. This is of course also true for pairwise alignments. In the pairwise case, none of the columns $Q \in \mathcal{C}(X, A) \setminus \mathcal{Q}'$ describe a (mis)match, i.e. they contain only insertions and deletions, while all $Q \in \mathcal{Q}'$ describe (mis)matches. A score-optimal alignment of two partial orders therefore corresponds to a maximum common induced subgraph (MCIS) of two transitive acyclic graphs. In both specifications of the problem, the scoring function will of course depend on the labels. We refer to Bunke (1997) for a discussion of the relationships of edit distances and maximum common subgraph problems in a more general setting.

4.3 Composition of Alignments

In order to study the composition of alignments it seems natural to first consider the properties of parts of given alignments. The most natural starting point is to consider restrictions induced by considering subsets of the input sequences. The following result, which generalizes Lemma 1 of W. Otto et al. (2011), provides a convenient starting point.

Lemma 8 *Let $(X, A, \prec)$ be an alignment and let $Y \subseteq X$. Then the induced subgraph $(X, A)[Y]$ with the partial order $\prec$ restricted to the non-empty intersections $Q \cap Y$ $\forall Q \in \mathcal{C}(X, A)$ is again an alignment. Furthermore, if $(X, A, \prec)$ satisfies* (A5)*, then the restriction to $(X, A)[Y]$ again satisfies* (A5)*.*

Proof 10 *Every induced subgraph of a complete graph is again a complete graph, hence* (A1) *holds for $(X, A)[Y]$, hence the connected components of $(X, A)[Y]$ are exactly the non-empty intersections of Y with the components Q of (X, A). Condition* (A2) *remains unchanged by the restriction to Y. Finally, the partial order $\prec$ satisfying* (A3) *restricted to the non-empty intersections $Q \cap Y$ for $Q \in \mathcal{C}(X, A)$ is a partial order that obviously still satisfies* (A4) *since the restriction to Y only removes some of the conditions in* (A4)*.*

To see that the restriction of $(X, A)[Y]$ again satisfies (A5) *it suffices to recall that the partial order in the column is given by $P \cap Y \prec Q \cap Y$ whenever $P \prec Q$ and both $P \cap Y \neq \emptyset$ and $Q \cap Y \neq \emptyset$. If one of the intersections is empty, axiom* (A5) *becomes void since the empty set is not a column in $(X, A)[Y]$. On the other hand, if the two restricted columns have entries (a, i) and (a, j) in the same row, then* (A5) *for $(X, A, \prec)$ ensures $(a, i) \prec_a (a, j)$, i.e., the implication* (A5) *remains true for the restricted alignment.*

Note that additional partial orders on connected components of the induced subgraph $(X, A)[Y]$ may exist that are not obtained as restrictions of the partial order on $\mathcal{C}(X, A)$. The reason is that omitting parts of the columns may allow a relaxation of their mutual ordering.

Rooted trees can be seen as partially ordered sets, with the natural partial order defined by $x \prec y$ if y lies on the unique path connecting x and the root of the tree. This special

case is thus covered in the general framework outlined here. Usually, tree alignments are defined on rooted *oriented trees*, however, where the relative order of siblings is preserved (T. Jiang et al., 1995; Höchsmann et al., 2004; Berkemer et al., 2017b), thus imposing additional restrictions on valid alignments. We will return to this point in some generality in the discussion section.

The fact that alignments are again totally or partially ordered sets implies that one can also meaningfully define alignments of alignments. As before, we start from a collection of finite partial orders $(X_a, \prec_a)$, $a \in S$. Let $\mathfrak{P}$ be a non-trivial partition of the rows, i.e., of S, whose classes we will write as S_α indexed by α. We write $X_\alpha := \bigcup_{a \in S_\alpha} X_a$. By construction, $X_\alpha \cap X_\beta = 0$ for $\alpha \neq \beta$, i.e., the site sets of the row classes are disjoint. The row partition $\mathfrak{P}$ thus implies a partition of X.

Lemma 9 *Let $(X, A, \prec)$ be an alignment of the $(X_a, \prec_a)$, $a \in S$, $\mathfrak{P}$ be a non-trivial partition of S, $X_\alpha := \bigcup_{a \in S_\alpha} X_a$ the site set of the row calls α and $(X, A, \prec)[X_\alpha]$ the corresponding sub-alignment of $(X, A, \prec)$. Then $(X, A, \prec)$ is isomorphic to the (vertex) disjoint union of the $(X, A, \prec)[X_\alpha]$ for all row classes α, augmented by extra edges (x', x'') whenever there is a column $Q \in \mathcal{C}(X, A)$ with $x' \in Q \cap X_\alpha$ and $x'' \in Q \cap X_\beta$ for classes $\alpha \neq \beta$.*

Proof 11 *By Lemma 8, the alignments $(X, A, \prec)[X_\alpha]$ are subalignments of $(X, A, \prec)$ and thus $(X, A)[X_\alpha]$ is an induced subgraph of (X, A). Their disjoint union therefore lacks exactly all edges that connect pairs of vertices that are in the same connected component of (X, A) but do not belong to the same class of rows α. Since the partial order on the columns of $(X, A)[X_i]$ is the one inherited from $(X, A, \prec)$, the re-composition of the columns also recovers the original partial order.*

A corresponding example is shown in Figure 31 where the alignment $(X, A, \prec)$ is composed out of sub-alignments $(X, A, \prec)[X_\alpha]$ and $(X, A, \prec)[X_\beta]$ with Q, Q_α, and Q_β as examples for connected components of the alignment graphs and their composition by disjoint union and extra edges (dashed lines) between elements of distinct sub-alignments.

The $(X, A, \prec)[X_\alpha]$ can also be interpreted as partially ordered sets whose *points* are the non-empty restrictions $Q \cap X_\alpha$ of the connected components of (X, A) to the row classes α.

Definition 34 (Quotient Graph)
We denote by $(X, A)/\mathfrak{P}$ the quotient graph whose vertices are the columns of the induced sub-graphs $(X, A)[X_\alpha]$, that is, the non-empty sets $Q \cap X_i$ where Q is a connected component of (X, A). Its edges are the pairs $(Q \cap X_\alpha, Q \cap X_\beta)$ for which both $Q \cap X_\alpha$ and $Q \cap X_\alpha$ are non-empty.

The connected components of the graph $(X, A)/\mathfrak{P}$ are therefore of the form $Q' := Q/\mathfrak{P} = \{Q \cap X_\alpha | Q \cap X_\alpha \neq \emptyset\}$. Note that Q' is non-empty since the column Q of (X, A) contains at least one element, which belongs to $(X, A)[X_\alpha]$ for at least one of the classes α of $\mathfrak{P}$. Thus there is a 1-1 correspondence between the connected components of (X, A) and those of $(X, A)/\mathfrak{P}$. The columns of $(X, A)/\mathfrak{P}$ therefore naturally inherit the partial order $\prec$ of $\mathcal{C}(X, A)$. We write $(X, A, \prec)/\mathfrak{P}$ for the quotient graph with this partial order on its connected components.

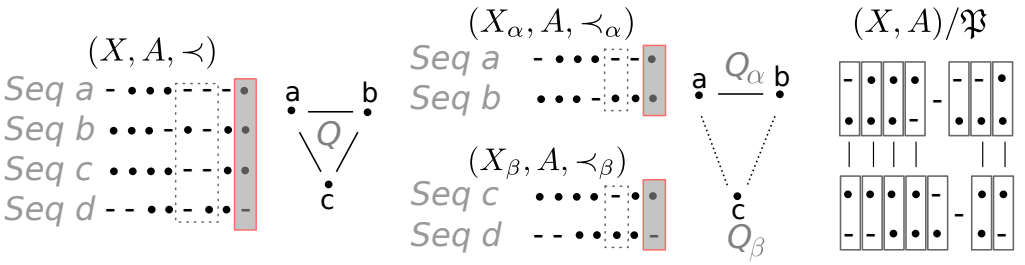


Figure 31: Example of an alignment (left) being composed out of sub-alignments (middle) corresponding to the partition of the rows in the two classes $S_\alpha = \{a, b\}$ and $S_\beta = \{c, d\}$. Columns marked by dashed lines show how the creation of sub-alignments removes gap columns. Column $Q \in \mathcal{C}(X, A)$ (marked in grey) is highlighted as an example. On the right, the two sub-alignments with the corresponding restrictions of Q are shown: $Q_\alpha \in \mathcal{C}(X, A)[X_\alpha]$ and $Q_\beta \in \mathcal{C}(X, A)[X_\beta]$ are connected components and complete subgraphs of the sub-alignment graphs and can be composed to Q by applying the disjoint union and adding extra edges between all elements in Q that are in distinct sub-alignments thus Q_α and Q_β (dashed lines). Indices at nodes in the graph refer to the sequence the node is coming from. The alignment $(X, A)/\mathfrak{P}$ on the right is the alignment of the sub-alignments (X_α, A) and (X_β, A). Thus the nodes in the alignment graph are columns of the sub-alignments. Alignment edges show matched columns. The unmatched columns correspond to the columns marked by dashed lines in the alignments on the left and middle.

Lemma 10 *$(X, A, \prec)/\mathfrak{P}$ is an alignment.*

Proof 12 *Consider the quotient graph $(X, A)/\mathfrak{P}$. By construction, each column Q' is a complete graph and contains at most one node for each class of $\mathfrak{P}$ since it is the quotient of a column of $(X, A, \prec)$ w.r.t. $\mathfrak{P}$. Also by construction, we have $P' \prec Q'$ for the columns of $(X, A)/\mathfrak{P}$ whenever $P \prec Q$ in $(X, A, \prec)$. Since there is a 1-1 correspondence between columns of $(X, A, \prec)$ and $(X, A, \prec)/\mathfrak{P}$, $\prec$ also serves as a partial order on the columns of $(X, A)/\mathfrak{P}$, which is by construction consistent with the partial order on $(X, A)[X_\alpha]$ for each of the row classes α.*

Theorem 3 *Let $(X, A, \prec)$ be an alignment and let $\mathfrak{P}$ be an arbitrary row partition. Then $(X, A, \prec)$ is isomorphic to the alignment $(X, A, \prec)/\mathfrak{P}$ of its restrictions $(X, A, \prec)[X_\alpha]$ to the row classes α of $\mathfrak{P}$.*

Proof 13 *Since $(X, A, \prec)/\mathfrak{P}$ is well defined by Lemma 10, Lemma 9 shows that expanding the classes points of $(X, A, \prec)/\mathfrak{P}$ into corresponding sets Q_α building the union of those that belong to a column of $(X, A, \prec)/\mathfrak{P}$ exactly recovers the columns of $(X, A, \prec)$ and their partial order.*

We note that the constituent alignments $(X_\alpha, A_\alpha, \prec_\alpha) := (X, A, \prec)[X_\alpha]$ have at most the same number of columns since "all gap" columns, $Q' = Q \cap X_i = \emptyset$, are removed. The decomposition of Theorem 3 can be applied recursively until each constituent alignment is one of the input posets $(X_a, \prec_a)$, $a \in S$. Any such recursive composition is naturally

represented as a rooted tree $\mathfrak{T}$. The leaves of $\mathfrak{T}$ are the input posets $(X_a, \prec_a)$, while the root represents $(X, A, \prec)$. Each internal node of $\mathfrak{T}$ corresponds to an alignment of its children. In particular, one can choose $\mathfrak{T}$ to be any binary tree.

The reverse of this type of decomposition underlies all *progressive alignment* schemes. One starts from a guide tree $\mathfrak{T}$ whose leaves are the $(X_a, \prec_a)$ and for each inner node of $\mathfrak{T}$ constructs an alignment (or a set of alternative alignments) from the (set of) alignments attached to its children. It is important to note that a score-optimal alignment $(X, A, \prec)$ in general is **not** the score-optimal alignment $(X, A, \prec)/\mathfrak{P}$ of score-optimal constituents $(X_\alpha, A_\alpha, \prec_\alpha)$, or, in other words, if $(X, A, \prec)$ is score-optimal, there is no guarantee that there is any nontrivial partition of the rows $\mathfrak{P}$ such that all the restrictions $(X, A, \prec)[X_i]$ are score-optimal subalignments. Progressive alignment methods thus cannot guarantee an exact solution of the multiple alignment problem. Results in practical applications depend substantially on the choice of the guide tree $\mathfrak{T}$. It has been suggested early (Feng and R. F. Doolittle, 1987), that $\mathfrak{T}$ should closely resemble the evolutionary history of the input sequences. Usually $\mathfrak{T}$ is constructed from distance or similarity measures between all pairs of input sequences – and usually pairwise alignments are employed to obtain these data. A special case of progressive alignment adds a single sequence in each step, instead of also considering alignments of larger sub-alignments.

4.4 Blockwise Decompositions

On the other hand, we can also decompose alignments into blocks of columns. More precisely, we consider an alignment $(X, A, \prec)$ and a partition $\mathfrak{Q} := \{Y_1, \ldots, Y_q\}$ satisfying the following properties:

(i) If $P \in \mathcal{C}(X, A)$ then $P \subseteq Y_k$ for some class $Y_k \in \mathfrak{Q}$.

(ii) There is a partial order $\lhd$ on $\mathfrak{Q}$ such that for any two distinct classes $Y_k, Y_l \in \mathfrak{Q}$ such that $Y_k \lhd Y_l$ whenever there are columns $P \in Y_k$ and $Q \in Y_l$ with $P \prec Q$.

We call the classes of such a partition *blocks*. By Lemma 8 each block $(X, A, \prec)[Y_k]$, $Y_k \in \mathfrak{Q}$ is again an alignment.

Theorem 4 *Given blocks $(X, A, \prec)[Y_k]$ with $Y_k \in \mathfrak{Q}$, and the partial order $\lhd$ on the blocks, there is an alignment $(X, A, \prec_\lhd)$, where $\prec_\lhd$ is an an extension of $\prec$ defined by $P \prec_\lhd Q$ if and only if $P \prec Q$ for $P, Q \in Y$ for some $Y \in \mathfrak{Q}$ and $P \prec_\lhd Q$ for $P \in Y_k$ and $Q \in Y_l$ with $Y_k \lhd Y_l$ and $k, l \in (1, q)$.*

Proof 14 *Each alignment block consists of the disjoint union of alignment column(s), thus the disjoint union of complete subgraphs. Given the partial order of alignment columns $P \prec Q$, this order is preserved inside the alignment blocks $Y_k \in \mathfrak{Q}$ as each block is an alignment, too. Given an alignment block Y with $P \prec Q$ for $P, Q \in Y$ for some $Y \in \mathfrak{Q}$, one can decompose this into two blocks Y_k and Y_l with at least one column in each block such that $P \in Y_k$ and $Q \in Y_l$. Based on the decomposition of Y into Y_k and Y_l one can restore the order of the alignment blocks such that $Y_k \lhd Y_l$ based on Y. Thus, one gets the order of $P \prec_\lhd Q$ that is present for the alignment columns P and Q as well as for the alignment blocks Y_k and Y_l.*

In the case of totally ordered inputs, the restriction $X_a \cap Y$ of a block Y to an input X_a is an interval of X_a and the columns in Y form an interval of the columns of $(X, A, <)$. Similarly, one can restrict the choice of blocks in such a way that $\lhd$ just "mirrors" the initial partial order, i.e., $Y_k \lhd Y_l$ if and only if $P \prec Q$ for P in Y_k and Q in Y_l, in which case $\prec_\lhd = \prec$ and the original alignment is recovered by the concatenation of the blocks. In particular, this also guarantees that valid block decompositions can be constructed for alignments satisfying (A5).

Each alignment can thus be recursively decomposed into blocks. This sets the stage for Divide-and-Conquer algorithms such as `DCA` (Stoye et al., 1997), which cuts the sequences to be aligned into subsequences and then concatenates the subalignments so as to optimize a global score. In order to find the best cut-points, the algorithm recurses on differently cut subsequences. Algorithms such as `dialign` (Morgenstern, 1999) work in a conceptually similar manner but use a bottom-up instead of a top-down approach: they first identify blocks with high sequence conservation as "anchors" and recurse to construct alignments for sequences between them.

An extreme case of the block-wise decomposition is to consider the division of an alignment $(X, A, \prec)$ into a single maximal (or minimal) alignment column P, and the rest $(X \setminus P, A', \prec)$ of the alignment. In order for $X \setminus A \lhd P$ to hold, we have to ensure that $p_a \not\prec_a q_a$ for all $p_a \in P$ and $q_a \in X \setminus P$, i.e., the column P must entirely consist of suprema of the respective input posets. Under this condition, we obtain a recursive column-wise decomposition of alignments. As we shall see in the following section, this recursion can also be used constructively.

4.5 Recursive Construction

Given a poset $(Y, \prec)$ we say that $P \subseteq Y$ is a *bottom set* if, for all $p \in P$, every $p' \prec p$ satisfies $p' \in P$. By definition, the empty set, Y itself, as well as the set $\{p' \in Y | p' \preceq y\}$ for each $y \in Y$ are bottom sets. Note, however, that P also may contain points that are incomparable to all other elements of P. Denote by $\sup P$ the set of *suprema* of P, i.e., the points such that there is no $p' \in P$ with $p \prec p'$. Clearly, if P is a bottom set and $p \in \sup P$ then $P \setminus \{p\}$ is again a bottom set. The latter observation suggests that there is a recursive construction for the set of alignments.

For simplicity of exposition, we first consider the pairwise case, i.e., the set of alignments of two finite posets $(X_1, \prec_1)$ and $(X_2, \prec_2)$. Denote by $\mathfrak{A}_Q^P$ the set of all pairwise alignments on bottom sets P in X_1 and Q in X_2. An alignment $\mathbb{A} \in \mathfrak{A}_Q^P$ is necessarily of one of three types:

(i) $\mathbb{A} = \mathbb{A}'\binom{p}{q}$ with $\mathbb{A}' \in \mathfrak{A}_{Q'}^{P'}$,

(ii) $\mathbb{A} = \mathbb{A}'\binom{p}{-}$ with $\mathbb{A}' \in \mathfrak{A}_Q^{P'}$, or

(iii) $\mathbb{A} = \mathbb{A}'\binom{-}{q}$ with $\mathbb{A}' \in \mathfrak{A}_{Q'}^P$,

where $P' := P \setminus \{p\}$ for $p \in \sup P$, $Q' := Q \setminus \{q\}$ for $q \in \sup Q$, and $\mathfrak{A}_\emptyset^\emptyset$ contains only the empty alignment.

The three cases correspond to (mis)match, insertion, and deletion. It is important to note that this recursion is in general not unique because the columns extracted from $\mathbb{A}$ in consecutive steps are not necessarily ordered relative to each other whenever $|\sup P| \geq 1$ or $|\sup Q| \geq 1$. It is, however, a proper generalization of the Needleman-Wunsch recursion (Needleman and Wunsch, 1970) for the pairwise alignment of ordered sets (strings): If the $\prec_a$ are total orders, then $\sup P_a$ always contains a single element, and we recover the usual Needleman-Wunsch algorithm. In order to have a proper start and end case for the recursion and thus DP algorithm, it is convenient to introduce "virtual" source and sink nodes being connected to all start or end nodes of the poset, respectively.

This idea generalizes to alignments of an arbitrary number of partial orders in the obvious way. Denote by $\mathfrak{A}(P_1, P_2, \ldots, P_N)$ the set of all alignments where the P_a are a bottom set of $(X_a, \prec_a)$.

Theorem 5 *Every alignment $\mathbb{A} \in \mathfrak{A}(P_1, P_2, \ldots, P_N)$ is of the form $\mathbb{A}'\Xi$ where the alignment column Ξ is a supremum w.r.t the partial order of $\prec$ of alignment columns and $\mathbb{A}' \in \mathfrak{A}(P'_1, P'_2, \ldots, P'_N)$. The column Ξ contains in row a either a gap row a, in which case $P'_a = P_a$, or $p_a \in \sup P_a$, in which case $P'_a = P_a \setminus \{p_a\}$, and does not entirely consist of gaps. For every column Υ of $\mathbb{A}'$ we have either $\Upsilon \prec \Xi$ or Υ and Ξ are incomparable.*

Proof 15 *The P'_a are again bottom sets, hence $\mathbb{A}'$ is an alignment. By assumption, there is a partial order on the columns $\prec$ of $\mathbb{A}'$. Since every non-gap entry in Ξ is a $p_a \in \sup P_a$, it follows that this partial order extends to $\mathbb{A}$ if and only if Ξ is a supremum, i.e., it is either incomparable with or larger than any column in $\mathbb{A}'$. Now suppose that the column Ξ contains a $q_a \notin \sup P_a$, i.e., there is a $p_a \in X_a$ with $p_a \succ q_a$. Consider the column Υ containing p_a. Then either no partial order $\prec$ on the columns exists (contradicting that $\mathbb{A}'$ is an alignment), or $\Upsilon \succ \Xi$ (contradicting that Ξ is a supremum for the alignment columns.*

The bottom sets are of course uniquely defined by their suprema. Clearly $\sup P$ is an antichain, i.e., its elements are pairwisely incomparable. Conversely, every antichain U in $(X_a, \prec_a)$ uniquely defines a bottom set $P := \{p \in X_a | p \preceq U\}$. It is obvious therefore that for two bottom sets P and Q it holds that $P = Q$ if and only if $\sup P = \sup Q$. Hence there is a 1-1 correspondence between the antichains of a partial order and their bottom sets. The recursion in the theorem can be written in terms of the antichains of the $(X_a, \prec_a)$. Note that the recursion of Theorem 5 can be transformed into an exact dynamic programming algorithm for alignment of posets, provided the scoring function is the sum of column-wise contributions.

In order to capture the more restrictive notion of alignments satisfying (A5) the recursion has to be modified in a such a way that for every (mis)match between two rows it can be ensured that all previously formed columns are either comparable in both rows or incomparable in both rows. This is non-trivial because this information is not purely local. For ease of discussion, we only consider the case of aligning two posets. There are at least two strategies to maintain this information.

Attempting to construct a similar recursion as in the (A4) case, one could store with each pair $P \in X_1$ and $Q \in X_2$ also all the set $\mathcal{M}$ of all matchings $\binom{p}{q}$ "to the right" of P and Q, i.e., $p \in X_1 \setminus P$ and $q \in X_1 \setminus Q$. Then every allowed matching/column $\binom{p'}{q'}$,

$p' \in \sup P$ and $q' \in \sup Q$ must satisfy: for all $\binom{p}{q} \in \mathcal{M}$ holds: either $p' \prec p$ and $q' \prec q$, or both p', p and q', q are incomparable. Every such pair can be appended to $\mathcal{M}$, with corresponding updates $P \to P \setminus \{p'\}$ and $Q \to Q \setminus \{q'\}$. Insertions and deletions of course only require the removal of either p' from P or q' from Q, respectively. Initially, $P = X_1$, $Q = X_2$, and $\mathcal{M} = \emptyset$. Every set of valid partial alignments is characterized by a triple $(P, Q, \mathcal{M})$.

An alternative approach is to store instead for each $p \in P$ and $q \in Q$ also the sets $c_Q(p)$ and $c_P(q)$ that can form matches $\binom{p}{q'}$, $q' \in c_Q(p)$ and $\binom{p'}{q}$, $p' \in c_P(q)$, respectively. Initially, we have $P = X_1$, $Q = X_2$, $c_Q(p) = Q$ for all $p \in P$ and $c_P(q) = P$ for all $q \in Q$. Whenever an alignment is continued with a (mis)match $\binom{p}{q}$, $p \in \sup P$, $q \in \sup Q$, we have to remove all candidates from $c_P(q')$ and $c_Q(p')$ that are inconsistent with $\binom{p}{q}$. That is: if $q' \prec q$, then $c_P(q') \leftarrow \{p' \in c_P(q') | p' \prec p\}$. If q and q' and incomparable, then $c_P(q') \leftarrow \{p' \in c_P(q') | p', p \text{ incomparable}\}$. The $c_Q(p')$ are updated correspondingly. In the case of an insertion $\binom{p}{-}$, we only need to remove p from $f_P(q')$, $q' \in Q$. Similarly, $\binom{-}{q}$ implies that q has to be removed from the $f_Q(p')$ for all $p' \in P$. We suspect that an encoding of alignment sets of the form $(P, f_Q : P \to 2^P; Q, f_P : Q \to 2^P)$ will be efficient if the poset has only small antichains. A more detailed analysis of this kind of recursive construction from the point of view of algorithmic efficiency will be considered elsewhere.

The POA algorithm (C. Lee et al., 2002) computes the alignment of two posets satisfying (A5), albeit with the restriction that one of the two inputs is totally ordered. This removes all ambiguities in the totally ordered poset and implies that, given any match $\binom{u}{v}$ in the alignment, all preceding matches $\binom{u'}{v'}$ satisfy $v' < v$ in the totally ordered set and thus u' must be a predecessor of u. The alignment thus must follow a single path in the Hasse diagram of the unrestricted input poset.

The recursive formulation of the poset alignments is an extension of the well-known Needleman-Wunsch alignment algorithm. Beyond many implementations of this algorithm, the implementation based on `ADPfusion` (Algebraic Dynamic Programming with compile-time fusion of grammar and algebra) (Höner zu Siederdissen, 2012) is designed in a way to be extendable to different scoring functions, problem descriptions, and data structures (Höner zu Siederdissen et al., 2015a). Future work thus will include the adaptation of the `ADPfusion` framework written in a functional language (Haskell) to the data structure of posets. Earlier adaptations of the Needleman-Wunsch algorithm to trees, forests and sets already exist (Berkemer et al., 2017b; Höner zu Siederdissen et al., 2015b) as described in Chapter 3.

4.6 Alignments as Relations

Pairwise alignments have a particularly simple structure. In particular, they are bipartite (undirected) graphs, and hence can be regarded equivalently as symmetric binary relations $R \subseteq X_1 \times X_2$. More precisely, we can identify a relation R with an undirected graph with vertex set $X_1 \dot\cup X_2$ and (undirected) edges $\{x_1, x_2\}$ whenever $(x_1, x_2) \in R$. We write this graph as $(X_1 \dot\cup X_2, R)$.

Relations have a natural composition. For $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ is defined by

$$(x, z) \in S \circ R \quad \text{iff} \quad \exists y \in Y \text{ s.t. } (x, y) \in R \text{ and } (y, z) \in S \tag{4.4}$$

In the following we will be interested in the following properties of binary relations:

(M) $(x, y) \in R$ and $(x, z) \in R$ implies $y = z$ and $(x, z) \in R$ and $(y, z) \in R$ implies $x = y$.

(P') There is a partial order $\prec$ on R such that $u \prec_1 x$ or $v \prec_2 y$ implies $(u, v) \prec (x, y)$.

(P) If $(x_1, y_1) \in R$ and $(x_2, y_2) \in R$ then $x_1 \prec x_2$ if and only if $y_1 \prec y_2$.

Lemma 11 *The composition of two binary relations satisfying* (M) *and* (P) *is again a binary relation satisfying* (M) *and* (P).

Proof 16 *Suppose $(x, z) \in R \circ S$. Then there is y such that both $(x, y) \in R$ and $(y, z) \in S$. By* (M)*, there is no other $y' \neq y$ with $(x, y') \in R$ and no $z' \neq z$ such that $(y', z') \in S$, hence in particular there is no $z' \neq z$ such that $(x, z') \in R \circ S$. Analogously, one argues that there is no $x' \neq x$ such that $(x', z) \in R \circ S$. Thus $R \circ S$ again satisfies* (M).

Suppose $(x_1, z_1), (x_2, z_2) \in R \circ S$. By (M) *there are unique vertices y_1 and y_2 such that $(x_1, y_1), (x_2, y_2) \in R$ and $(y_1, z_1), (y_2, z_2) \in S$, respectively. Now suppose $x_1 \prec_1 x_2$. Then* (P) *implies $y_1 \prec_2 y_2$, and using* (P) *again yields $z_1 \prec_3 z_2$. Starting from $z_1 \prec_3 z_2$, the same argument yields $z_1 \prec_1 z_2$. Conversely, suppose $(x_1, z_1), (x_2, z_2) \in R \circ S$ and x_1, x_2 are incomparable. By* (M) *there are unique vertices y_1 and y_2 with $(x_1, y_1), (x_2, y_2) \in R$ and $(y_1, z_1), (y_2, z_2) \in S$, for which (P) now implies that they are incomparable. Using the same argument again shows that that z_1 and z_2 also must be incomparable. Hence concatenation preserves not only the relative order but also comparability, i.e., $R \circ S$ again satisfies* (P).

It is easy to see that Axiom (P') is in general not preserved under concatenation: Requiring only (P') allows the intermediate vertices y_1 and y_2 to be incomparable. Hence it is possible in this scenario to have $x_1 \prec_1 x_2$, incomparable vertices y_1 and y_2, and $z_2 \prec_3 z_1$ with $(x_1, y_1), (x_2, y_2) \in R$ and $(y_1, z_1), (y_2, z_2) \in S$ while the concatenation violates the (P').

A relation satisfying (M) and (P') can easily be extended to an alignment $(X_1 \cup X_2, R)$ considering each edge (x_1, y_1) and considering all unmatched positions, i.e., every $\{x'\}$ such that there is no $y \in X_2(x', y)$ and every $\{y'\}$ such that there is no $x \in X_1(x, y')$ as alignment columns. The relative order of these columns is inherited from the partial order $(X_1, \prec_1)$ and $(X_2, \prec_2)$.

Lemma 12 *Every pairwise alignment satisfying* (A1)*,* (A2)*,* (A3)*, and* (A4) *can be written as an extension of the a binary relation $R \subseteq X_1 \times X_2$ satisfying* (M) *and* (P')*. Conversely, every binary relation $R \subseteq X_1 \times X_2$ satisfying* (M) *and* (P') *gives rise to an alignment satisfying* (A1)*,* (A2)*,* (A3)*, and* (A4).

Proof 17 *By definition, all edges are incident to one vertex in X_1 and one vertex in X_2, thus the graph is a bipartite matching. Condition* (M) *is therefore equivalent to* (A1) *and* (A2) *for the case of two input posets. Axiom* (A3) *implies the ordering required by* (P') as well as its extension to the in/del columns. (A4) and (P') equivalently guarantee the existence of the partial order on the columns that satisfy (A3).

Theorem 6 *Every pairwise alignment satisfying* (A5) *corresponds to a binary relation $R \subseteq X_1 \times X_2$ satisfying* (M) *and* (P).

Proof 18 *Axiom* (A5) *simplifies to* (P) *in the case of only two inputs. The existence of the required partial order on the set of all columns is guaranteed by Lemma 7.*

This suggests that the more restrictive condition (A5) may be a more natural condition for defining alignments of partially ordered sets. As a down-side, however, it seems that there is no convenient recursive construction of the search space similar to the dynamic programming approaches for sequence alignment. Instead, it seems more natural to treat this class of alignment problems as maximum induced subgraph problems.

Composition of binary relations is a powerful tool to construct multiple alignments. Suppose we are given a set of posets $(X_a, \prec_a)$ and a set $\mathcal{R}$ of pairwise relations satisfying (M) and (P) such that the graph representation of $\mathcal{R}$ is tree, then there is a unique multiple alignment satisfying (A5) obtained as the transitive closure of the graph on X with edges defined by the $R \in \mathcal{R}$. However, not every alignment can be represented in this manner. As a simple counterexample consider the alignment of the three sequences `a,b` and `c`:

```
                                        composition
(a,b,c)          (a,b)      (b,c)      ((a,b),(b,c))
a  A-C           a  A-C                   a -A-C
b  -BC           b  -BC     b  -BC        b --BC
c  AB-                      c  AB-        c A-B-
```

The first column gives an alignment of three input strings `a`, `b`, and `c`. It contains the pairwise alignments `(a,b)` and `(b,c)`. Their relational composition (shown in the last column) contains only the matches `BB` from `(b,c)` `CC` from `(a,b)`. The match `AA` from the original alignment is not reconstructed in the composition of the pairwise alignment. One easily checks that, by symmetry, no composition of pairwise alignments recovers the original 3-way alignment. Hence not all alignments can be represented as relational compositions of pairwise alignments. On the other hand the progressive approach, in which sequence `c` is aligned to the pairwise alignment of `a` and `b` yields the example alignment. In fact, Lemma 10 implies that in principle every alignment can be obtained by a progressive alignment scheme. If $\mathcal{R}$ contains cycles, then there is no guarantee that the transitive closure $\widehat{A}$ of $\bigcup_{R\in\mathcal{R}} R$ is an alignment: In general, both conditions (A1) and (A2) will be violated. So-called *transitive alignment* approaches deliberately accept this at an intermediate stage. Various heuristics can be used to remove superfluous edges from the graph $(X, \widehat{A})$, that is they construct a subgraph (X, A), $A \subseteq \widehat{A}$ that again satisfies all conditions of a valid alignment.

4.7 Tree Alignments

Alignments on trees and forests are described in detail in Chapter 3. This section will shortly summarize the principles from the point of view of generalized alignments and compositional properties.

A forest $T = (V, E)$ is defined based on a pair of vertex set V and edge set E and two mutually exclusive order relations on the elements in V: the ancestor order $\prec$ and the sibling order $\lhd$ as described in Subsection 2.2.1. Consider a forest T with vertex set V and define T_v with vertex set $V \setminus \{v\}$ as follows: (1) if v is the root of a subtree, delete v

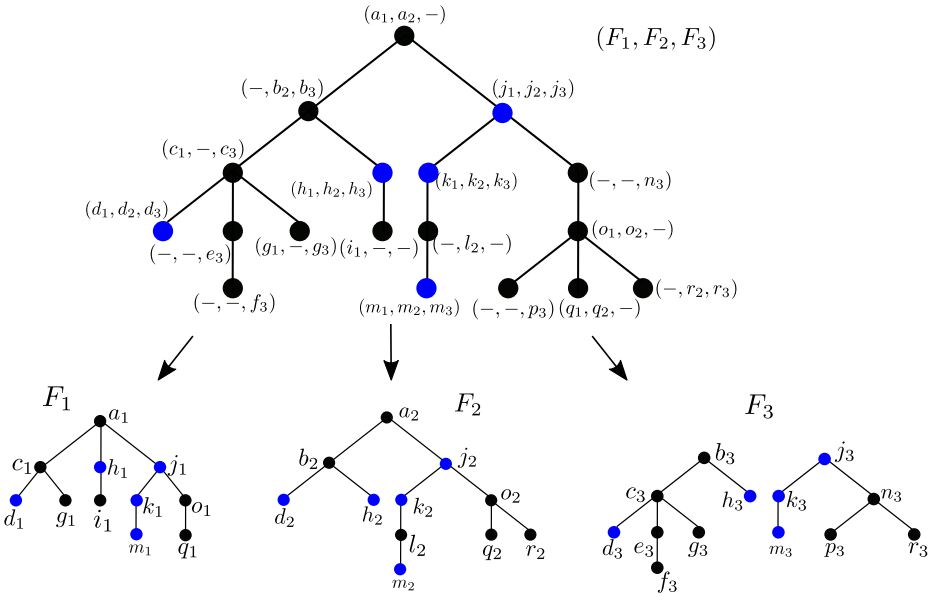


Figure 32: Example of a forest alignment of three forests (bottom). The resulting forest (top) is the superstructure combining all of the input trees. The nodes labels correspond to alignment columns and blue nodes indicate matches such that they exist in all the input trees. Original trees can be recovered from the supertree by only taking nodes without gap symbol in the corresponding alignment column. A node with gap symbol is then removed and its edges contracted such that its children will be its parents children afterwards. This can be seen in F_1 where node b does not exist and nodes c_1 and h_1 become children of the root a_1.

and replace the tree $T(v)$ rooted at v by trees rooted at the children of v in sibling order; (2) if v is not the root of a subtree, contract the edge from the parent of v to v. That is, the children of v become children of the parent of v. It is not hard to check that both the ancestor and sibling orders for T_v is simply the restriction of $\prec$ and $\lhd$ to $V \setminus \{v\}$.

A forest alignment is defined as a forest T such that each vertex v is labeled by an alignment column Q_v. The constituent tree T_s, $s \in S$ is obtained from T by first simplifying the label on T to $Q_v \cap X_s$ at each vertex v; then all v with $Q_v \cap X_s = \emptyset$ are removed by deletion or contraction of their parent edge as outlined above (T. Jiang et al., 1995; Höchsmann et al., 2004; Berkemer et al., 2017b). Thus T_s has the vertex set $V' := \{v \in V | Q_v \cap X_s \neq \emptyset\}$ and both its ancestor and sibling orders are the restriction of $\prec$ and $\lhd$ to V'. An example for an alignment of three forest structures is shown in Figure 32. Tree or forest alignments thus fit seamlessly into the mathematical formal for partial order alignments. We simply have to require that the alignment graph (X, A) satisfies (A1) and (A2) and that properties (A3) and (A5) hold w.r.t. *both* partial orders $\prec$ and $\lhd$. This observation suggest how alignments satisfying an analog of (A5) can be defined in a meaningful way for a much broader class of discrete structures.

A notion of alignment similar to tree/forest alignments is used in computational biology

for RNA structures (see also Subsection 2.3.2), where base pairs need to be preserved in addition the total order of the input sequences (Möhl et al., 2010). Here, however, only consistency similar in flavor to (A4) is enforced, suggesting that it may also be of interest to relax the requirement that restriction to the columns Q for which $Q \cap X_a \neq \emptyset$ exactly recovers the input tree $(X_a, \prec_a, \lhd_a)$.

4.8 Alignments of Graphs

In Section 4.2 we have seen that alignments of partially ordered sets can alternatively be viewed as alignments of graphs from a very restricted class, namely transitive acyclic digraphs. This begs the question whether the construction can be generalized to arbitrary (di)graphs. In this section we consider an input set of digraphs G_a, $a \in S$, with vertex sets $V(G_a) = X_a$ and edge sets $E(G_a)$, respectively. As before, we write $X = \bigsqcup V(G_a)$, introduce a set of alignment edges A, and denote by $\mathcal{C}(X, A)$ the set of connected components of the (undirected) graph (X, A).

Definition 35 (Graph Alignment)
A triple (X, A, E^*), where A is a set of unordered pairs on X and E^* is a relation on $\mathcal{C}(X, A)$, is a multiple alignment of the graphs G_a, $a \in S$, where A if the following conditions are satisfied:

(G1) *$Q \in \mathcal{C}(X, A)$ is complete subgraph of (X, A).*

(G2) *If $(a, i) \in Q$ and $(a, j) \in Q$, then $i = j$.*

(G3) *If $(a, i) \in P$, $(a, j) \in Q$ for some $P, Q \in \mathcal{C}(X, A)$ and $((a, i), (a, j)) \in E(G_a)$ then $(P, Q) \in E^*$*

(G4) *If $(P, Q) \in E^*$ then there is a row a with $(a, i) \in P$, $(a, j) \in Q$ and $((a, i), (a, j)) \in E(G_a)$,*

(G5) *If $(P, Q) \in E^*$, $(a, i) \in P$, and $(a, j) \in Q$ then $((a, i), (a, j)) \in E(G_a)$.*

Condition (G4) is redundant and is included here only to emphasize the similarity to the constructions in the previous sections. It may also be interesting to consider graph alignments that satisfy only (G4) but not (G5).

Lemma 13 $(\mathcal{C}(X, A), E^*) \simeq (X, \bigcup_{a \in S} E(G_a))/\mathcal{C}(X, A)$.

Proof 19 *The vertex set $X/\mathcal{C}(X, A)$ has a single representative for each column $Q \in \mathcal{C}(X, A)$. By axioms* (G3) *and* (G5)*, there is an edge $(P, Q) \in E^*$ if and only there $(a, i) \in P$ and $(a, j) \in Q$ with $((a, i), (a, j)) \in E(G_a)$ for some $a \in S$. The edge set on the r.h.s., amounts to identical condition.*

Thus $(\mathcal{C}(X, A), E^*)$ is obtained from $(X, \bigcup_{a \in S} E(G_a))$ by identifying the vertices within each alignment column. In particular, therefore, the set $\mathcal{Q}'$ of columns Q such that $Q \cap X_a \neq \emptyset$ for all a in a given subset $S' \subseteq S$ forms an induced subgraph $(\mathcal{C}(X, A), E^*)$ that is present in each G_a. Observation 4 thus remains true for graphs in general:

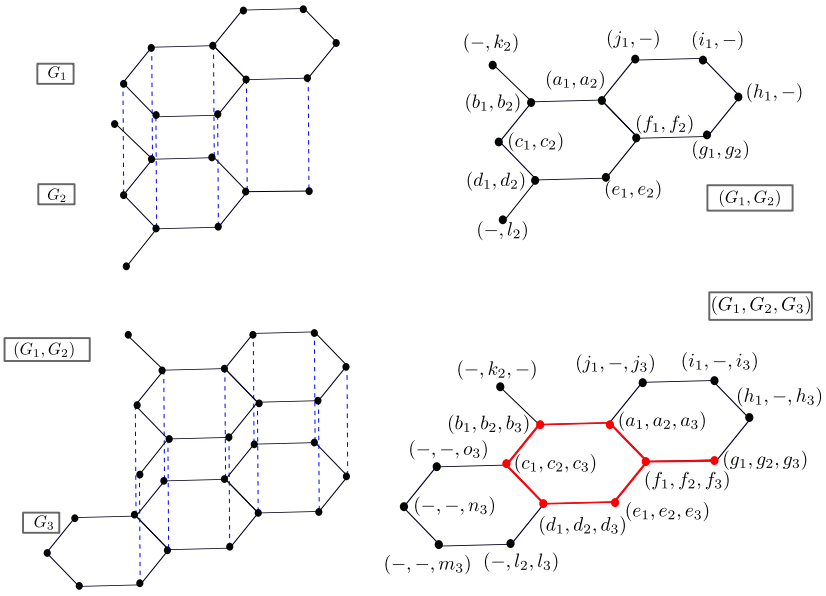


Figure 33: Example for (progressive) graph alignment of G_1 and G_2 (top) with aligned graph structure on the r.h.s and alignment of (G_1, G_2) with G_3 and aligned graph structure again on the r.h.s. Dashed blue lines show matches between nodes of the input graphs. Labels at nodes correspond to alignment columns, indices refer to input graphs G_1, G_2 or G_3. The red subgraph is the maximal common induced subgraph of all three input graphs.

Observation 5 *Let (X, A, E^*) be an alignment of graphs (X_a, E_a), $S' \subseteq S$ a subset of columns, and $\mathcal{Q}' \subseteq \mathcal{C}(X, A)$ such that $X_a \cap Q \neq \emptyset$ for all $a \in S'$ and $Q \in \mathcal{Q}'$. Then the graph with vertex set $\mathcal{Q}'$ and edge set E^* is an induced subgraph of (the graph representation of) (X_a, E_a).*

We note in passing that alignments of ordered and partially ordered sets assuming axiom (A5) are special cases of the graph alignments satisfying (G5), since total and partial orders are isomorphic to transitive acyclic digraphs. One easily checks that (G3) and (G5) indeed reduce to the corresponding statements for the (partial) orders.

Again this is in particular true for pairwise alignments. Given two graphs G_1 and G_2 and a common induced subgraph H (strictly speaking together with an embedding of H into G_1 and G_2) the graph defined by identifying the copies of H in G_1 and G_2 is the pairwise alignment $G_1 \bullet_H G_2$ of the input graphs. Naturally, an optimization criterion will be used in practice. The problem of aligning graphs therefore coincides with the maximum common induced subgraph (MCIS) problem. Finding MCIS is well known to be a NP-complete problem and closely related to the maximal common edge subgraph problem (MCES), together often referred to as the maximal common subgraph problem (MCS) (Ehrlich and Rarey, 2011; Duesbury et al., 2018). However, several approaches exist to find exact or approximate solutions for connected (cMCS) or disconnected (dMCS)

common subgraphs using different algorithmic strategies such as backtracking algorithms, dynamic programming, or clique-finding.

It is very easy to check that Lemmas 8, 9 and 10 – and thus also Theorem 3 – remain true for the graph alignments of Definition 35. Indeed, the alignment of two graphs is again a graph. Its vertices, corresponding to the columns of the alignment, are labeled by the content of the columns. Therefore, we can build alignments of alignments for graphs. In particular, furthermore, progressive alignments of graphs are well-defined. Given a guide tree $\mathcal{T}$, at each inner node of $\mathcal{T}$ the maximum common induced subgraph of the graphs at its child-nodes is computed, and the graphs are "glued together" at the common vertices. An example for the alignment of graphs is shown in Figure 33.

It is important to note that graph alignment in the sense used here – namely requiring a matching between vertices and notion of structural congruence between the alignment and its constituent graphs – are more restrictive than some concepts of "graph alignments" discussed in the literature. In particular, we make a sharp distinction here between "graph alignments" and various approaches of comparison by means of graph editing, see e.g. Emmert-Streib et al. (2016) for a recent review.

4.9 Alignments for General Structures

So far, we have considered alignments for sequences (strings), partially ordered sets, rooted ordered trees, and graphs. How far can we generalize the idea of alignments, and what are minimal conditions for well-defined alignments? Let us start from a finite space $(X, \mathscr{S})$ with some structure $\mathscr{S}$. We are not really interested in the particular properties of $\mathscr{S}$. Examples for $\mathscr{S}$ might be systems of not necessarily binary relations, topologies, proximities, etc. As a minimum requirement we ask that $(X, \mathscr{S})$ admits well-defined subspaces, that is, if $Y \subseteq X$, then there exists a unique subspace $(Y, \mathscr{S}_Y) =: (X, \mathscr{S})[Y]$. Furthermore we require that

$$(X, \mathscr{S})[Z] = ((X, \mathscr{S})[Y])[Z] \tag{4.5}$$

holds for all $Z \subseteq Y \subseteq X$, i.e., that induces subspaces that can be formed stepwisely in a consistent manner. This property is satisfied for the examples we have considered so far: strings and totally ordered sets in general, partial orders, as well as directed and undirected graphs. It also holds for ternary relations such as betweenness, as well as topologies, proximities, and similar constructions.

Now suppose we are given input spaces $(X_a, \mathscr{S}_a)$ for all $a \in S$. As in the previous sections, we set $X := \bigcup\!\!\cdot_{a \in S} X_a$, we introduce a set A of edges connecting the vertices in X and write $\mathcal{C}(X, A)$ for the set of connected components of the graph (X, A). Furthermore, we define

$$\mathcal{C}_a := \{Q \in \mathcal{C}(X, A) | Q \cap X_a \neq \emptyset\}.$$

Endowing $\mathcal{C}(X, A)$ with some structure $\mathscr{S}$ consider the subspace $(X, \mathscr{S})[\mathcal{C}_a]$ obtained from $(X, \mathscr{S})$ to the connected components (columns) of (X, A) in which X_a is represented. As in the previous sections we assume

(X1) $(X, A)[Q]$ is a complete graph for all $Q \in \mathcal{C}(X, A)$, and

(X2) $|X_a \cap Q| \leq 1$ for all $a \in S$ and $Q \in \mathcal{C}(X, A)$.

Assumption (X1) implies that there is a 1-1 correspondence between the columns of $Q \in \mathcal{C}_a$ and the elements $q \in X_a$ define by $Q \cap X_a = \{q\}$. Denote the corresponding map by $\pi_a : \mathcal{C}_a \to X_a$. The condition that "projecting" $(\mathcal{C}(X, A), \mathscr{S})$ down the constituent rows $a \in S$ recovers the input spaces can then be expressed as

(X3) $(\mathcal{C}(X, A), \mathscr{S})[\mathcal{C}_a] \simeq (X_a, \mathscr{S}_a)$ with π_a being an isomorphism.

This construction provides a well-defined notion of an alignment in a very general setting. Again, the restriction of the alignment to a set $\mathcal{C}'$ of columns that are represented in X_a for all $a \in S'$, i.e., $(\mathcal{C}(X, A), \mathscr{S})[\mathcal{C}']$ is a common subspace of the $(X_a, \mathscr{S}_a)$ with $a \in S'$. This corresponds the poset alignments satisfying (A5).

Properties (X1), (X2), and (X3) are sufficient to ensure that key properties of totally ordered alignments still hold in this much more general setting. Repeating the simple arguments leading to Lemmas 8, 9 and 10 above, we observe:

(i) The restriction $(X, A, \mathscr{S})[Y]$ to $Y \subseteq X$ is an alignment for the restricted input spaces $(X_a, \mathscr{S}_a)[X_a \cap Y]$.

(ii) If $\mathfrak{P}$ is a partition of X into groups of rows, the quotient $(X, A, \mathscr{S})/\mathfrak{P}$ is an alignment of alignments: The rows of $(X, A, \mathscr{S})/\mathfrak{P}$ are of the form $(\mathfrak{C}(X, A), \mathfrak{S})[\mathcal{C}']$, where $\mathcal{C}' := \{C \in \mathcal{C}(X, A) | C \cap X_a \neq \emptyset, a \in S'\}$ where $S' \subseteq S$ determines a class of the row-wise partition $\mathfrak{P}$. That is, every row of $(X, A, \mathscr{S})/\mathfrak{P}$ is (isomorphic to) a subspace of $(\mathfrak{C}(X, A), \mathfrak{S})$.

(iii) For a given class of $\mathfrak{P}$ determined by the row indices, we observe that by construction the restriction of $(X, A, \mathscr{S})[Y]$ to $Y := \bigcup_{a \in S'} X_a$ is isomorphic to $(\mathfrak{C}(X, A), \mathfrak{S})[\mathcal{C}']$. By assumption,
$(\mathcal{C}(X, A), \mathscr{S})[\mathcal{C}_a] = ((\mathcal{C}(X, A), \mathscr{S})[\mathcal{C}'])[\mathcal{C}_a]$ for all $a \in S'$. Therefore we can construct $(X, A, \mathscr{S})$ as the alignment $(X, A, \mathscr{S})/\mathfrak{P}$ of the alignments $(X, A, \mathscr{S})[Y]$ of the rows in each class of the partition $\mathfrak{P}$.

We conclude therefore, that alignments defined by (X1), (X2), and (X3) can be decomposed recursively into alignments of alignments on all spaces with subspaces satisfying Equation 4.5. In particular, these properties are sufficient to guarantee that progressive alignments are well defined.

A natural question that arises at this abstract level is whether for any collection $(X_a, \mathscr{S}_a)$, $a \in S$, there exists an alignment. To answer this question we consider trivial alignments for which $A = \emptyset$. Then every alignment column contains an element from exactly one of the X_a. Thus there is a 1-1 correspondence between $\mathcal{C}(X, \emptyset)$ and X, ensuring that $(X, \emptyset, \mathcal{S})$ and $(X, \mathcal{S})$ are isomorphic. By (X3), $(X, \emptyset, \mathcal{S})$ is an alignment of the $(X_a, \mathscr{S}_a)$ whenever $(X, \mathscr{S}^*)[X_a] \simeq (X_a, \mathscr{S})$ for all $a \in S$. The existence of such a "disjoint union" $(X, \mathscr{S}^*)$ is thus a sufficient condition for the existence of alignments. All the examples discussed in this section allow such "disjoint unions" and hence support alignments of arbitrary input data.

4.10 Concluding Remarks

In this chapter, the compositional properties of sequence alignments have been analyzed and explored up to the generalization to much more general structures. It has been shown that meaningful concepts of alignments are not restricted to ordered sets as inputs, but can be extended to very general relational or topological structures that need not bear any resemblance with order relations. The key property of the generalized alignments considered here is that the restriction of the alignment to a row recovers the input row. While this property is a simple consequence for the familiar sequence alignments, it becomes an important defining property of alignments in general. It suffices under very mild conditions of the structure of input spaces to ensure that alignments of alignments and recursive, row-wise decompositions of alignments are well-defined. We have observed, furthermore, that some well-studied examples of alignment problems, such as tree alignment and the alignment of totally ordered sets to a poset seamlessly fit into the framework developed here.

Part III

Bioinformatics Applications of Dynamic Programming

CHAPTER 5

On Popular Input Data to Dynamic Programming Algorithms

Contents

One area where dynamic programming (DP) algorithms are often used is bioinformatics. This chapter gives a short overview of basic principles in molecular biology conceptionalized to the main underlying assumptions in bioinformatics research. Even though DP algorithms are applied in other areas such as computer science and linguistics (Wagner and Fischer, 1974; Levenshtein, 1966; Kondrak, 2000; Cysouw and Jung, 2007; Steiner et al., 2011), this chapter focuses on applications in bioinformatics and corresponding data analysis.

While DP algorithms are widely used in bioinformatics applications, not every problem can be solved using dynamic programming. Thus, the analysis is usually composed out of a set of algorithms and self-implemented scripts and programs. Towards the end of the chapter, popular algorithms and methods used in bioinformatics are shortly described (Section 5.4). For more detailed explanations and descriptions, see for example Alberts et al. (2008) concerning the biological background, and Zvelebil and Baum (2007) and Lemey et al. (2009) regarding the area of bioinformatics.

Analyzing biological data usually includes several distinct DP algorithms where each of them requires input in a certain format. Even though various data formats are known and widely used in bioinformatics, there are still slight differences and preferences to be considered. Thus, setting up appropriate data structures and formats is an important step before starting the analysis. Here, necessary information has to be stored in a way to provide easy access and reformatting possibilities when needed during the analysis. This requires a basic knowledge of the data and its origins.

5.1 Biological Sequences

Genetic information is encoded and transmitted via sequences in the cell. The following subsections will describe the structure of DNA and RNA sequences as well as proteins, i.e., amino acid sequences.

5.1.1 DNA

The deoxyribonucleic acid (DNA) molecule is a directed polymer which is composed of four different nucleobases that are connected via a sugar(Deoxyribose)-phosphate backbone. Nucleobases connected to the sugar-phosphate backbone are called *nucleotides*. The four different bases are: adenine (A), cytosine (C), guanine (G) and thymine (T) as shown in Figure 34. Always two bases are complementary to each other and can form *base pairs* using hydrogen bonds. Base pairs are formed out of A and T or C and G, where G-C pairs are connected by three and A-T pairs by two hydrogen bonds. Two antiparallel strands of DNA form a double helix by base pairs build out of complementary bases. As strands are running in opposite directions, they are usually called sense and antisense or forward and backward strand. Stacking interactions of nucleotides in distinct strands stabilize the double helix, see Figure 35. Deoxyribose molecules are connected to each other via the 3'C atom of one molecule to the 5'C atom of the subsequent molecule. Thus, the DNA strand has a 5'end and a 3'end and is used to indicate directions. The forward strand is directed from 5'end to 3'end.

The DNA double helix is well protected and efficiently packed into chromosomes. In archaea and eukarya, DNA is additionally wrapped around *histones* (see Figure 34). The

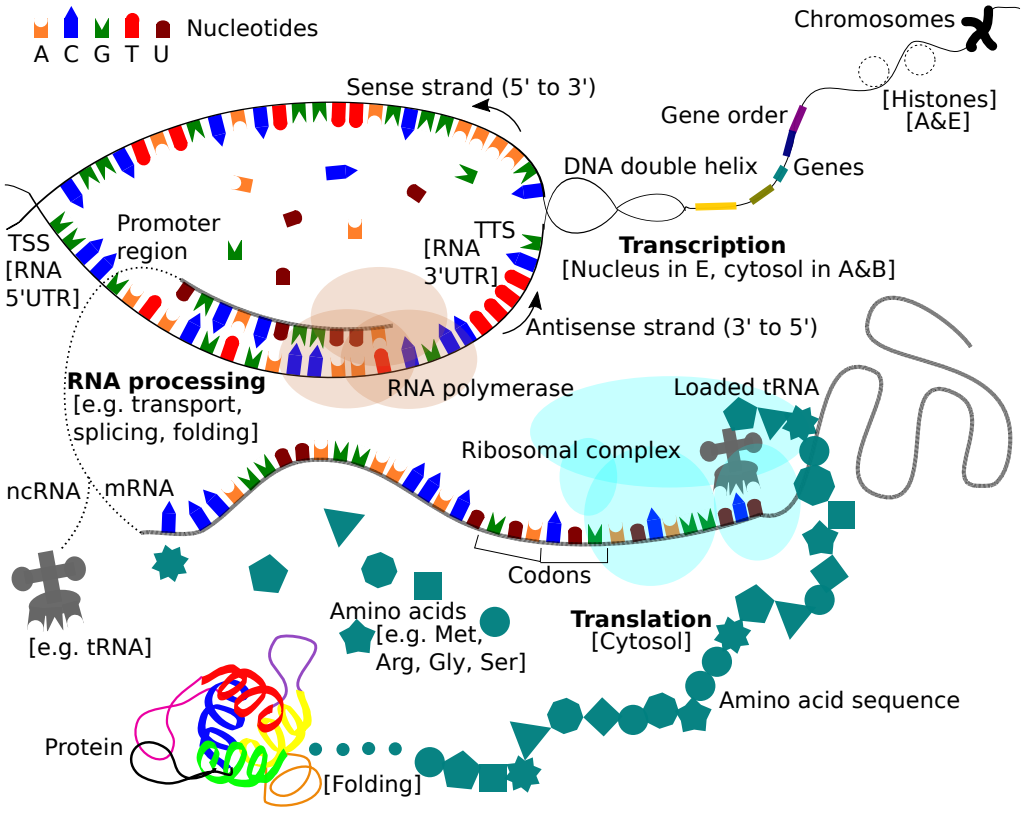


Figure 34: Schema showing the principles of transcription and translation. Additional information are shown in squared brackets including details about occurrences in archaea (A), bacteria (B) or eukarya (E).

complex formed by histone molecules and the DNA is called *nucleosome*. During cell division, the complete genetic material of a cell is duplicated and distributed into the daughter cells. The process of DNA duplication by DNA polymerases is called *replication*. As it is only possible to copy DNA from a single strand, the double helix is partially opened such that the DNA polymerase can attach to a single strand and add a complementary version. Single pieces of copied DNA are ligated together in order to retrieve a completely copied genome (Alberts et al., 2008).

Another way of reading the DNA sequence is called *transcription*. Here, RNA polymerases attach to one strand of the DNA and transcribe the DNA sequence into a complementary RNA sequence (see next subsection for more details on RNA). The DNA is transcribed using RNA polymerases and transcription is initiated by a transcription start site (TSS) located in the promotor region and terminated by usually at least one transcription termination site (TTS). Chapter 8 will go into more detail about transcription termination in archaea.

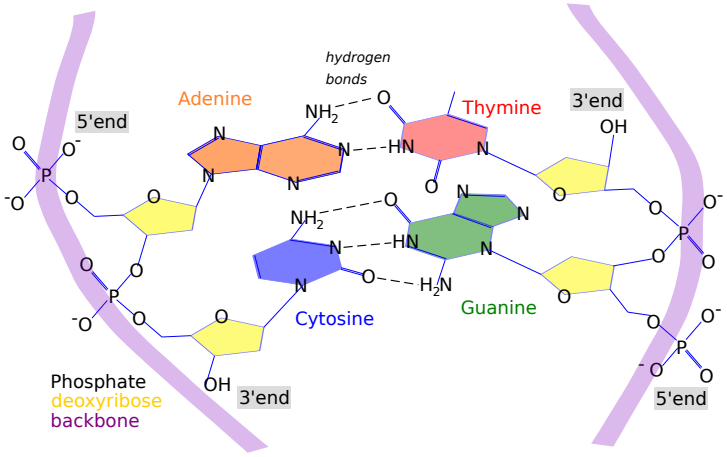


Figure 35: Schematic figure showing the structure of DNA. The backbone consists of phosphate-deoxyribose chains where nucleobases are connected. The double helix is stabilized due to stacking of aromatic rings in the nucleobases and hydrogen bonds formed in between complementary bases.

Regions on the DNA strands that are functional entities are called *genes* where the order of genes, called *synteny*, can have effects on gene regulation and expression. More about syntenic relations of genes will be explained in Chapter 6.

5.1.2 RNA

Transcription (see Figure 34) is done similar to DNA replication as the double helix is opened and RNA polymerase is attached to the single strand. After transcription initiation, the RNA polymerase creates a nucleotide sequence complementary to the current DNA strand. However, instead of thymine (T) Uracil (U) is used, a nucleotide similar to thymine differing by one methyl group. The product is a ribonucleic acid (RNA) sequence, a single strand molecule consisting of the bases A, C, G and U. The production of the RNA molecule is called elongation phase. The last phase is called termination phase where RNA polymerase is detached from the DNA template and RNA sequence and transcription stops (Alberts et al., 2008).

The resulting RNA sequence is processed further in order to stabilize it against digestion. As eukaryotic cells contain a nucleus, RNA sequences need to be transported into the cytosol. In opposite, prokaryotic cells do not contain a nucleus and transcription and translation both take place in the cytosol. Further processes on the RNA include splicing of introns or folding of sequences into certain structures.

The complete set of RNA in a cell is called *transcriptome*. RNA sequences are divided into two main groups: coding RNAs (messenger RNA (mRNA)) that encode for proteins (see the following subsection for more details on proteins) and the group of non-coding RNA (ncRNA) molecules that are not translated into proteins but form certain structures and fulfill different tasks in the cell (Mattick and Makunin, 2006).

A well-known example are tRNAs that are part of the ribosomal complex and translate between nucleotide codons and amino acids (see also Figure 34 and the section about tRNAs below). Also rRNAs are part of the ribosomal complex during translation. The ncRNAs that are responsible for nucleotide modifications such as methylations are called small nucleolar RNA (snoRNA). Other ncRNAs are part of gene regulating processes such as micro RNA (miRNA) and long non-coding RNA (lncRNA). Further RNA processing steps involve small nuclear RNA (snRNA) and Y RNA molecules (Santosh et al., 2014) which are described below. While some ncRNAs exist in all organisms, most types of ncRNAs are restricted to subsets of the tree of life.

5.1.3 Proteins

The *translation* process from the mRNA into a protein is initiated by ribosomal proteins and ribosomal RNA (rRNA) molecules forming the ribosomal complex (Alberts et al., 2008). Based on the mRNA sequence, a corresponding amino acid sequence is synthesized.

Specific transfer RNA (tRNA) molecules have a binding site, called *anticodon*, where three adjacent bases of the tRNA bind to three adjacent bases in the mRNA, called *codon*. The loaded tRNA additionally carries an amino acid corresponding to its (anti)codon which is then transferred to the amino acid chain currently synthesized in the ribosomal complex as depicted in Figure 34. Amino acids are composed out of different side groups that can form bonds such that the sequence of amino acids folds into a certain structure. The structure is essential for the function and stability of the protein (Alberts et al., 2008).

There are 22 different amino acids in biological systems, even though many more exist. Proteins are the product of the translation from an mRNA sequence (see also Figure 34). The set of all proteins in a cell is called *proteome*. Similar to RNA sequences, proteins fold in certain structures that are essential for their function and stability. Special cases are enzymes that interact with other proteins, RNAs or further molecules using principles of the 'lock and key' model as only certain molecules match an enzyme's binding sites or pockets in order to initiate a reaction. Like RNAs, proteins play an important role in maintaining a cell's function. As depicted in Figure 34, proteins are part of the transcription and translation processes as polymerases as well as ribosomal proteins forming the ribosomal complex are proteins interacting with RNAs (Alberts et al., 2008).

Genes will be transcribed into RNA and some of them translated to proteins. Regions in the DNA that are translated are called *coding regions*. All other regions are referred to as *intergenic regions*. Transcription and translation processes are highly regulated by various mechanisms in the cell including small RNA molecules that can act as enhancer or inhibitor. Binding sites for them exist, e.g., in intergenic regions or the regions directly in front (upstream) or behind (downstream) of a gene. These regions are called untranslated region (UTR), thus 5' UTR (upstream region) and 3' UTR (downstream region) of the gene (Alberts et al., 2008).

5.2 The Phylogenetic Tree of Life

Evolution is the way of how life on earth emerged and evolved. Darwin (1859) mainly coined the concept of species evolution due to reproduction allowing errors and mutations

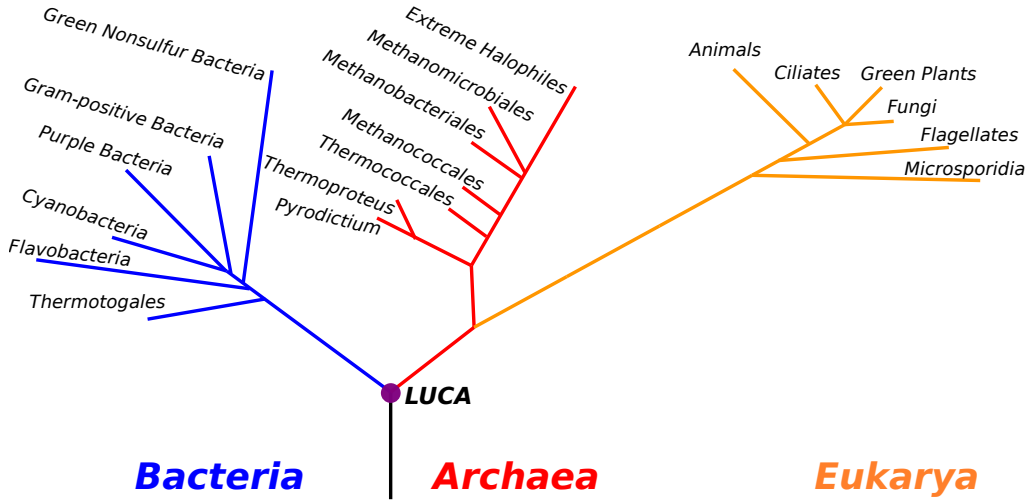


Figure 36: Phylogenetic tree of life as proposed by C. R. Woese et al. (1990) after discovering that prokaryotes evolutionary form two groups, archaea and bacteria. LUCA is the last universal common ancestor of Archaea, Bacteria and Eukarya, however, it is assumed that cells with genetic material have existed already before and origins of life occurred much earlier (Forterre and Gribaldo, 2007).

such that the offspring is significantly different from its parents. This leads to the possibility of adaptation to environmental changes or advantages towards organisms of the same or distinct species regarding reproductive fitness or acquisition of food and living space.

Evolutionary processes are usually depicted by the shape of a tree similarly to a family tree showing relatives of a family. In both cases, the root of the tree usually depicts the oldest member of the family and time passes towards the leaves of the tree. In case of evolutionary relationships between species, this tree is called phylogenetic tree. The following subsections will explain structure and events in phylogenetic trees used in bioinformatics research.

The current structure of the phylogenetic tree of life (TOL) includes three main groups of organisms, i.e. *domains*, bacteria, archaea and eukarya as depicted in Figure 36. Before the discovery of archaea (C. R. Woese et al., 1990), the tree consisted of only two groups, prokarya and eukarya. Archaea and bacteria still form the group of prokarya, however, despite many ongoing debates of the structure of the tree of life, they are now considered to form two evolutionary distinct domains inside the tree (Eme et al., 2017; Gribaldo et al., 2010). It is possible to find common traits between all three domains of life as well as clear differences. It is assumed that genes having orthologs in species spread all over the tree are well conserved and ancient and might have been part of the genome of the *last universal common ancestor*. However, horizontal gene transfer (HGT) events and incomplete data sets make it more difficult to infer phylogenetic relationships. The lowest universal common ancestor (LUCA) is assumed to be the last common ancestor of archaea, bacteria and eukarya. However, cells with genetic content might have existed before and

evidence exists that origins of life occurred much earlier (Forterre and Gribaldo, 2007).

This subsection shortly describes relationships between archaea, bacteria and eukarya including differences and commonalities. The focus is based on the transcription process as it will be described and analyzed further in Chapter 8. A short summary about current assumptions on LUCA and inference methods is given, introducing further results described in Chapter 7.

5.2.1 Archaea, Bacteria and Eukarya

Based on comparisons of 16S rRNA, C. R. Woese et al. (1990) discovered that prokaryotes can be grouped into two evolutionary distinct domains, archaea and bacteria. After the discovery, archaeal, bacterial and eukaryotic cells were target to comparative studies in order to find differences and commonalities and infer evolutionary relationships. Baumann et al. (1995) describe a few early considerations on archaea, bacteria and eukarya. Morphology and size are more similar for archaea and bacteria and they both lack a nuclei. Most of the prokaryotes have one circular chromosome, however, there is a large number of prokaryotic cells including additional plasmids varying in size and number (Baumann et al., 1995). Prokaryotic genomes are compact and many regulatory mechanisms are shared among archaea and bacteria.

However, processes related to transcription and translation seem to be more similar between archaea and eukarya (Baumann et al., 1995; Gribaldo et al., 2010), i.e. they share a significant amount of ribosomal proteins and structural characteristics of RNA polymerases. RNA modification processes such as splicing occur in all three domains, however in distinct versions. Self-splicing introns are shared among archaea and bacteria (Nawrocki et al., 2018) but enzymatic splicing occurs in archaea and eukarya (Watanabe and Yoshinari, 2013).

Since the archaeal clade was discovered after the distinction between eukaryotes and bacteria (C. R. Woese et al., 1990), most of archaeal traits are still unknown and research is based on comparative approaches to eukaryotic or bacterial counterparts. As stated above, archaeal RNA synthesis is generally considered to be more closely related to transcription in eukaryotes than to bacterial transcription. The archaeal RNA polymerase is similar to the eukaryotic RNA polymerase II, and the basal promoter elements in archaea are similar to their eukaryotic pendants (TATA box and BRE); general transcription factors TBP (TATA binding protein), TFB (transcription factor B) and TFE (transcription factor E) resemble the eukaryotic proteins TBP, TFIIB and TFEII, respectively (for a review see: Fouqueau et al. (2018)). Transcriptional regulators, however, seem to be more similar to those in bacteria (Bell, 2005). Thus, the archaeal transcription machinery consists of a mixture of bacterial-like and eukaryotic-like components. Transcription termination in archaea will be discussed in more detail in Chapter 8.

5.2.2 Evolutionary Aspects towards LUCA

A longstanding goal of evolutionary biology is to infer the traits of the most ancient organisms. Several criteria can be used to assess whether or not a particular gene sequence may have been in the lowest common ancestor (LCA). Conserved presence of a gene in a large number of archaea and bacteria can provide evidence of presence prior to the

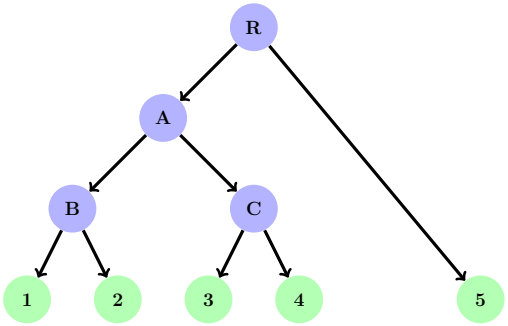


Figure 37: Example of a phylogenetic tree with inner nodes A, B and C, root node R colored blue and leaves $1, 2, 3, 4$ and 5 shown in green.

formation of taxonomically distinct bacteria and archaea, and if the phylogeny of the gene separates the domains as occurs with many ribosomal proteins and rRNA sequences, presence in the LCA - also called LUCA in this context - is predicted with greater confidence. Although molecular markers such as the 16s ribosomal DNA gene (C. R. Woese et al., 1990), some ribosomal proteins (Hug et al., 2016), and some nucleotide polymerase subunits such as RpoB (Case et al., 2007) have indicated overall species relationships upon phylogenetic analysis, molecular comparison of these molecules does not give insight into the metabolisms which power these polymerases.

A major challenge of reconstructing the physiology of the LUCA from gene phylogeny is derived from the metabolic diversity of life today; since very few genes - especially those coding for metabolic functions - are present in all organisms, it is difficult to assess which metabolic traits may have been early from the simple view point of conservation (Charlebois and W. F. Doolittle, 2004). Given the existence of HGT, phylogenetic scenarios cannot be reconstructed without further information, as also described in Subsection 5.3.1.

5.3 Genetic Evolutionary Relationships

Evolutionary relationships between genes are based on a common ancestral gene that evolved over time and was distributed into distinct species. Genetic evolutionary relationships were mainly coined by Fitch (1970), but also other works highly influenced the field (Ohno, 1970; Kimura et al., 1968; Nei, 1987; Zuckerkandl and Pauling, 1965). The set of evolutionary relationships between species or genes is called *phylogeny* and it can be displayed in a *phylogenetic tree* (Durbin et al., 1998) as explained in the following subsections.

As explained in Subsection 2.2.1, a tree structure is usually represented as a directed acyclic graph (DAG) where the edges are directed from the root towards the leaves. In a phylogenetic tree, this direction corresponds to time, thus the root node is the ancestor of all other nodes in the tree, and every inner node represents the ancestor of species or genes that are depicted at nodes below towards the leaves of the tree. This is depicted in Figure 37, where blue nodes are inner nodes and green nodes are the leaves.

Regarding evolutionary history, ancestral genes or species might be unknown. To trace back the history of a certain gene or species, one therefore depicts evolutionary events at the inner nodes of the phylogenetic tree which might have caused the genes or species to evolve into to distinguishable entities, displayed at the children nodes of the inner node.

The *lowest common ancestor* (LCA) of two or more nodes in the tree is the lowest inner node of the tree (closest to the leaves) that lies on the intersection of the leaves' paths to the root node. The LCA of nodes 1 and C in Figure 37 is node A and the LCA of nodes $1, 2$ and 5 is R.

5.3.1 Events in the Phylogenetic Tree

A phylogenetic tree displays evolutionary history of species (species tree) or genes (gene tree). Genes and genomes undergo editing processes called mutations caused either by replicating the genome during meiosis or mitosis or due to environmental influences acting on the cell or genome (Durbin et al., 1998; Lemey et al., 2009). Mutations introduce changes into the genetics of an organism and are the basic principle of evolution. Even though many changes might lead to lethal conditions, others may lead to abilities that help the organism gain advantages over its competitors (Darwin, 1859).

There are various events that can occur. If only a single element in the nucleotide sequence is affected, the mutation is called *point mutation*. As depicted in Figure 38 (left), this mutation can be the exchange of a base (substitution), its deletion or an insertion of an additional nucleotide. Point mutations can affect the folding of RNA molecules, binding sites or the translation process (Lemey et al., 2009). Point mutations in transfer RNA (tRNA)s are discussed in Chapter 6. Deletion and insertion events are summarized under the term indel, see also Chapter 2. During translation, a missing or inserted base affects the triplet code, shifting the open reading frame (ORF) which results in a different amino acid sequence. A nucleotide exchange might only change the amino acid corresponding to the given triplet. However, the genetic code is designed in a way that the exchange of nucleotides at the third position in a codon (wobble position) mostly encodes for the same kind of amino acid (Lemey et al., 2009). Point mutations that still encode for the same amino acid are called synonymous, otherwise they are called non-synonymous mutations (Lemey et al., 2009). A change in an amino acid can cause a change in the structure of the protein or its active binding site such that binding its target molecules becomes impossible but other molecules might be able to bind the altered binding site.

Not only single nucleotides are target to mutations but also complete genes or sections of the genome, alleles and even complete genomes. Figure 38 (right) shows mutations occurring on gene level.

Duplication events occur frequently (Koonin, 2005) and in different variations. Not only single genes are duplicated but also whole genomes or alleles. It is assumed that duplicated genes are prone to change of function (Ohno, 1970; Koonin, 2005). An example for duplicated genes are tRNAs, that are assumed to have evolved from one gene (Eigen et al., 1989) where a high sequence similarity was maintained over a long time span (Velandia-Huerto et al., 2016; Berkemer et al., 2017a). This is called concerted evolution and will be discussed in Chapter 6.

The *loss* or deletion of a gene is mostly a fluent process. Thus, a gene's sequence is modified step by step until it first looses its functionality and at some point, cannot be

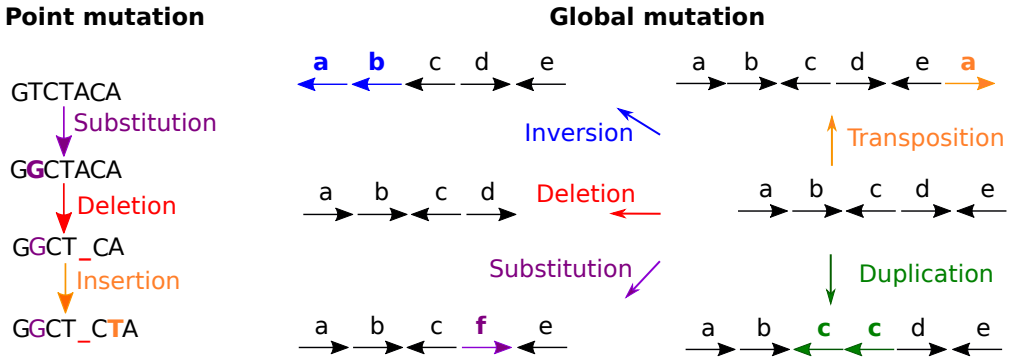


Figure 38: Point mutations (left) occur on single nucleotides whereas global mutations (right) can affect genes, sections of the genome, alleles or complete genomes. Arrows indicate genes located on the forward or reverse strand.

identified as the original gene anymore. An important pathway to gene loss is *pseudogenization*, which can in many cases be detected by means of sequence similarity to the target elements.

Remolding refers to an evolutionary event that changes the type or subtype of a molecule. The best-known examples are changes of the anticodons in tRNAs such that the tRNA now refers to a different amino acid (Rawlings et al., 2003; Sahyoun et al., 2015). Hence, given two tRNAs with distinct types but a high sequence similarity, the pair of tRNAs is reported as remolding event. By definition, remolding events can only be associated with duplication events.

The *insertion* of a (new) gene usually happens due to a remote duplication event, thus a gene being copied and inserted at a location further away from the original gene (*transposition*) or the substitution or inversion of genes as depicted in Figure 38.

Transposons are elements that are copied between remote positions in the genome and thus, often also appear as repetitive elements (McClintock, 1950) and remote duplication events.

Another way of duplicating genes is called *tandem duplication* which happens when neighboring gene(s) are duplicated resulting in several adjacent genes appearing twice.

The above mentioned evolutionary events all describe events during lateral gene transfer (LGT). However, especially in prokaryotes and in interaction with viruses, *horizontal gene transfer* can be found in many phylogenetic scenarios (Ochman et al., 2000). The event of HGT happens when genetic material is exchanged in other ways than through vertical reproduction, thus LGT, such as *Retrotransposition* which is the effect of copying a gene from a different genome, e.g. a virus, that is then inserted in the host cell's genomes (Bennett et al., 2008).

High numbers of mutations can cause the appearance of new species, thus parts of an existing species evolving in distinct directions. Then, the newly evolved species may still contain a similar set of genes, however, genes might have undergone mutational changes. This is called a *speciation* event.

Events and corresponding evolutionary relationships are depicted in Figure 39 and explained in more detail in Subsection 5.3.3.

5.3.2 Gene Families

Gene families are derived from common ancestral genes. Members of a gene family can occur as single genes or gene clusters and can be divided into multi-gene families or superfamilies (Ohta, 2001). Genes in a *multi-gene family* are derived from an ancestral gene that evolved due to a high amount of duplications. Duplicated genes that originated from the same gene keep a similar function but accumulate mutations such that different versions of the gene can be distinguished. However, their regulation is controlled by common mechanisms. Examples are tRNAs or rRNAs (Ohta, 2001).

A gene *superfamily* can contain single genes, gene clusters or multi-gene families whose regulatory controls might differ between subfamilies. An example is the globin superfamily with three subfamilies: the α-like cluster, the β-like cluster and the single gene myoglobin (Ohta, 2001).

Repetitive Elements

Repetitive elements are regions of the deoxyribonucleic acid (DNA) that occur in high frequencies throughout the genome. There exist various kinds of repetitive elements and functional purposes are often not known (Liao, 1999; Nei and Rooney, 2005; Sun et al., 2007; Nishihara et al., 2006). It is assumed that *repetitive elements* evolved by a high number of duplication events with occasional transpositions to other locations in the genome. This is called *concerted evolution*. There exist repetitive elements with important functions such as tRNA and Y RNA genes. As their distribution and evolutionary development is further discussed in Chapter 6, a short overview of these elements is given here. The development of repetitive elements is closely related to evolutionary events which will be explained in more detail in Subsection 5.3.1.

tRNAs. Transfer RNAs (tRNAs) originated before the separation of the three Domains of Life. There is clear evidence, furthermore, that all tRNA genes are homologs, deriving from an ancestral "proto-tRNA" (Eigen et al., 1989), which in turn may have emerged from even smaller components (Eigen and Winkler-Oswatitsch, 1981). They are indispensable in all organisms. In addition to their ancestral role as mediators of the genetic code (see e.g. Florentz et al. (2012)), tRNAs have secondarily acquired additional functions, reviewed e.g. in McFarlane and Whitehall (2009) and Soares and Santos (2017). Beyond *bona fide* tRNAs there is a rich universe of tRNA-derived repetitive elements (SINEs) (Sun et al., 2007) and of small RNAs that either directly derive from tRNAs (Rozhdestvensky et al., 2001; Iacoangeli et al., 2008) or arose indirectly as exapted SINEs (Nishihara et al., 2006).

Y RNAs. Like tRNAs, mammalian Y RNAs are pol III transcripts (O'Brien et al., 1993). They form the RNA component of Ro ribonucleoprotein (RoRNP) particles (Lerner et al., 1981; Hendrick et al., 1981). The molecules exhibit a characteristic structure that has been extensively studied in the past (Farris et al., 1999; Teunissen et al., 2000). They are essential for the initiation of chromosomal DNA replication in vertebrates (Christov et al., 2006), probably in conjunction with the Origin Recognition Complex (Kheir and Krude,

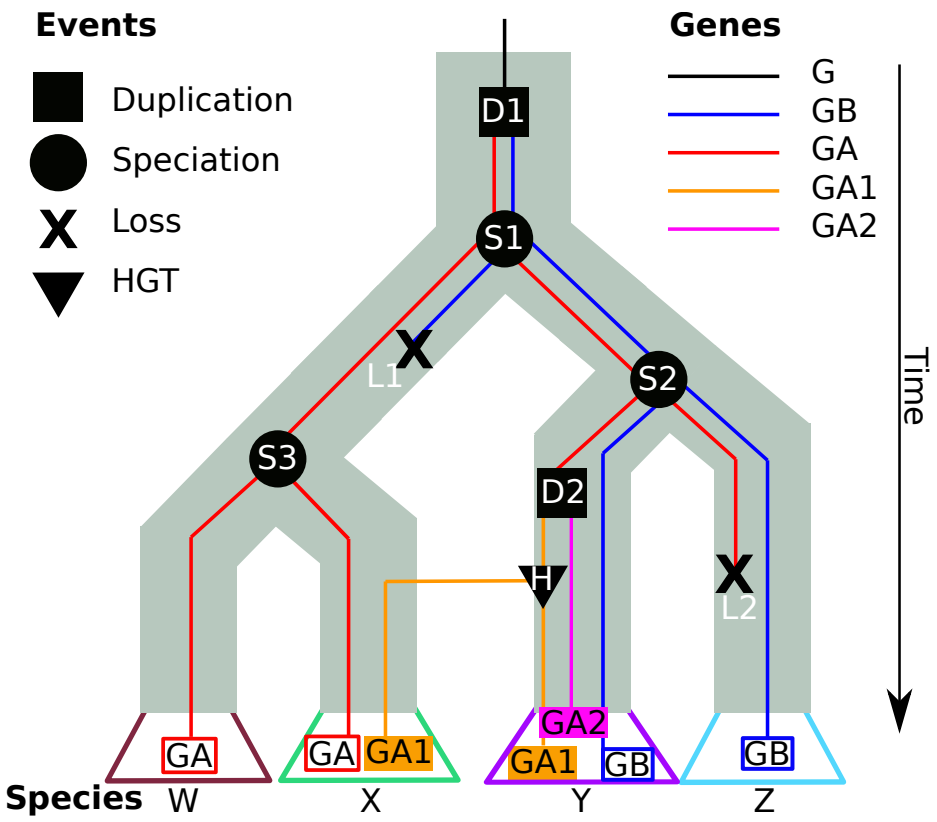


Figure 39: Schema showing a species tree (gray area) and evolution of gene G inside the species tree. Phylogenetic events on the tree are duplication (square), speciation (circle), loss (cross) and horizontal gene transfer (HGT, triangle). The tree contains four species W, X, Y and Z that have different sets of descendants of gene G. Phylogenetic relationships between genes inside species are explained in the text.

2017). As part of the RoRNP they are involved in RNA stability and cellular responses to stress (A. Hall et al., 2013). In addition, small Y RNA fragments are enriched in apoptotic cells (Rutjes et al., 1999).

5.3.3 Homology Relations

Evolutionary relationships between genes are usually defined based on the LCA of referred genes. If genes have a common ancestor, they are called *homologous*. Most important relationships will be summarized in the following, for a detailed review, see, e.g. Koonin (2005).

Paralogous genes can be found in the same species and appeared due to a duplication event at their LCA. Genes $GA1$ and $GA2$ in species Y shown in Figure 39 are paralogs due to the duplication event $D2$.

Two genes in distinct species whose LCA is a speciation event are called *orthologous*. Given the tree in Figure 39, genes GA in W and GA in X are orthologous with $S3$ as their LCA.

The event H in Figure 39 is an HGT event whose descendants are called *xenologs*, here gene $GA1$ in X and $GA1$ in Y.

Depending on the relative locations of speciation and duplication events, the term co-paralog is used to describe the relation between $GA1$ and GB in Y as well as $GA2$ and GB in Y. Analogously, genes $GA1$ in Y and GA in W are co-orthologs as well as $GA2$ in Y and GA in W.

Evolutionary relationships can be inferred based on the genetic sequences at the leaves of the tree. Thus, it is quite difficult to infer gene losses or HGT events without further information. Therefore, one might assume that genes $GA1$ and GA in X are paralogs even though their LCA is $S1$, a speciation event. In this case, they are called pseudo-paralogs. A similar situation exists for genes GA in W and GB in Z. Here, one might assume that they are orthologous, however, their LCA is duplication event $D1$. Thus, they are called pseudo-orthologs (Koonin, 2005).

Inferring Orthology Relations

A precise record of the history of a gene family, i.e., an accurate reconstruction of a phylogenetic gene tree, is an indispensable prerequisite for a detailed description of the functional evolution of its members and the assessment of innovations (Capra et al., 2013; Holland, 2013). The exact placement of gene duplication and gene loss events relative to a species tree is also of key importance in the context of forward genomics (Hiller et al., 2012). The first crucial step towards elucidating the history of a gene family is to distinguish orthologs, i.e., gene pairs that originated from a speciation event, from paralogs, which arose by gene duplication (Fitch, 1970) as described in Subsection 5.3.1.

As depicted in the phylogenetic tree in Figure 39, it is not always clear how genes evolved only considering sequences and mutual comparisons thereof. The number of tools to reconstruct evolutionary history from sequence data is growing, and inferring orthology relations between genes is done in various ways. A basic assumption about homologous genes is that they show a relatively high sequence similarity in comparison to other non-homologous genes (Lemey et al., 2009). However, distinguishing between orthologs and paralogs is still a problem and has large influences on the resulting topology of the phylogenetic tree. A short summary on existing methods to infer orthology relations and reconstruct phylogenetic scenarios is given in Subsection 5.4.5.

5.4 Algorithms & Methods

The sections above gave an introduction to biological data and corresponding backgrounds that frequently appear in data sets to be analyzed and explored in bioinformatics research. This sections shortly describes well-known algorithms in bioinformatics used to analyze biological data. The basic principles of sequence alignment and structure prediction have been described in Section 2.3 and will only be explained from the data perspective.

5.4.1 Alignments

As described in Subsection 2.3.1, sequence alignments are used to compare (biological) sequences. Adapted to various problems of sequence matching, there exist many extensions and variations. The extension to compare more than two sequences, called multiple sequence alignment (MSA) is described in Subsection 2.3.1 and Chapter 4. However, exact and optimal solutions to string matching problems might not give biologically relevant solutions or reflect true biological processes. Comparing only two sequences of very different length, one might want to create a *local alignment* such that the shorter sequence is a part of the longer one but long sequences of gaps around the shorter sequence are scored with a neutral score in order not to get an overall negative score. This algorithm is called *Smith-Waterman algorithm* (Smith and Waterman, 1981a; Smith and Waterman, 1981b).

A further adaptation of the alignment algorithm deals with the structure of gaps. Since a long connected stretch of inserted or deleted nucleotides is biologically more realistic than the same number of singleton gaps, sequence alignments with affine gap costs are frequently used in bioinformatics, called *Gotoh algorithm* (Gotoh, 1982). As described in Chapter 3, the algorithm is based on different scores for gap openings and gap extensions such that the algorithm prefers long connected gaps instead of many singletons.

Gaps display insertions and deletions that happened during evolutionary development of sequences. Another frequently occurring event is the duplication (see also Subsection 5.3.1). In order to reconstruct duplication events one can adapt the alignment algorithm to be able to detect longer stretches of matching positions, also called duplication alignments (Berkemer et al., 2017a; Velandia-Huerto et al., 2016), described in more detail in Chapter 6.

A popular tool to search for similar (homologous) sequences in large sequence data bases is called basic local alignment search tool (BLAST) (Altschul et al., 1990). The program and its various extensions are based on the dynamic programming (DP) algorithm of pairwise alignments. However, BLAST is equipped with various heuristics in order to efficiently find matching target sequences in a large set of sequence data.

5.4.2 Sequencing & Mapping

After the discovery of the first sequencing process (Sanger et al., 1977), sequencing methods and technologies started to quickly evolve. Sequencing DNA made it possible to compare genomic data of distinct organisms and get deeper insights into molecular processes. By now, it is also possible to sequence RNA (*RNAseq*), such that the current transcriptome of a cell can be explored and compared to other transcriptome data sets. With high-throughput sequencing methods (also Next Generation Sequencing (NGS)), it is possible to sequence large numbers of sequences at the same time leading to an increase of the amount of sequencing data. However, a drawback of early and current NGS sequencing is the increase of error rates with sequence length. Thus, sequence lengths are usually restricted to around 100 or 150 nucleotides. Such short sequences are fragments of DNA or ribonucleic acid (RNA) and are called *reads*. Sequencing a genome at first is called de novo sequencing, and is followed by assembling a puzzle of reads based on overlapping sequences (Myers et al., 2000) in order to retrieve the complete genome or transcriptome. However, after the successful assembly of an organism's genome, further

sequencing data of this organism can be assembled using the existing reference genome. This process is called *mapping* which simplifies the assembly of reads.

By mapping the reads to the reference genome, the number of reads matching a position in the genome, called *coverage*, can be determined. Using RNAseq data, the coverage of certain transcripts gives insights into transcription regulation and current processes of the cell (Z. Wang et al., 2009; Limbach and Paulines, 2017). Thus, next to various ways of library preparation for transcriptome sequencing, the in-silico analysis of reads is an important step. In most cases, reads are first cleaned from low quality reads and sequencing adapters and then mapped to a reference genome. As RNA molecules are usually further processed after transcription, such reads may not match the DNA directly which needs to be taken into account during mapping (Z. Wang et al., 2009; Costa-Silva et al., 2017; Canzar and Salzberg, 2017). Mapping procedures use dynamic programming approaches together with heuristics to reduce the search space.

Problems can be caused by repetitive transcripts mapping to several locations of the DNA (multi-mappers) or spliced transcripts where intron sequences are missing. This can be improved using paired-end data, which consists of pairs of reads that originate from the same transcript. This information is stored in the read data set and is used by the mapper to confirm placement of reads. Short MSAs or local alignments are further methods to gain more information about the reads and possible mapping positions (Canzar and Salzberg, 2017). The analysis of a specialized RNAseq data set and its implications for transcription termination is described in Chapter 8.

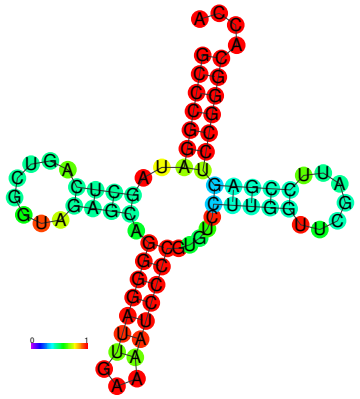
5.4.3 RNA folding

Structures of RNA molecules are an essential part of their functionality (Alberts et al., 2008). In order to be able to display an RNA *structure*, one defines the nucleotide sequence as the primary structure (one dimension, see Figure 40, top line), the base pairing pattern as the secondary structure (two dimensions, Figure 41a) and the structure in three-dimensional space as tertiary structure (see Figure 41b). Molecule complexes based on interactions of several RNA molecules, proteins or other elements, are called quaternary structure. RNA secondary structures are a common tool to work with. A frequently used visualization is the *dot-bracket notation*, where base pairs are notated as parentheses '(' and ')' and unpaired bases as dots '.', as depicted in Figure 40 (bottom line). For basic algorithmic principles of RNA structure prediction, see Subsection 2.3.2.

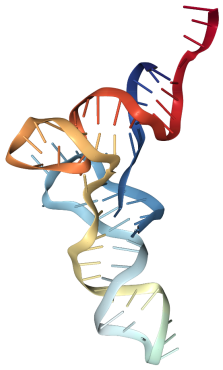
RNA folding is based on the stabilizing interaction of the nucleobases stacking onto each other when base pairs are formed, similarly to the stacking in the DNA double helix. The energy of structural features such as stacked base pairs, hairpin loops or multi loops is supposed to be independent of context and can be measured or computed. The minimization of these energies allows to calculate for the stability of an RNA molecule and probable folding conformations (Hofacker et al., 1994; Zuker and Stiegler, 1981).

```
5' GCCCGGAUAGCUCAGUCGGUAGAGCAGGGGAUUGAAAAUCCCCGUGUCCUUGGUUCGAUUCCGAGUCCGGGCACCA 3'
   (((((((..((((........)))).(((((((...)))))))).....(((((.......)))))))))))....
```

Figure 40: Primary structure (sequence) and secondary structure in dot-bracket notation of $tRNA^{Phe}$. Further structures are depicted in Figures 41a and 41b.



(a) Secondary structure of $tRNA^{Phe}$.



(b) Tertiary structure of $tRNA^{Phe}$.

Figure 41: Structures of a $tRNA^{Phe}$. Figure 41a describes its secondary structure, created using `RNAalifold`, a program of the `ViennaRNApackage` (Bernhart et al., 2008; Lorenz et al., 2011b). Figure 41b depicts the tertiary structure, taken from the PDB with PDB ID 3BBV (`http://www.rcsb.org/structure/3BBV`) (L. Jiang et al., 2009; Berman et al., 2000). Corresponding sequence in Figure 40.

Figures 40, 41a and 41b show sequence, secondary structure and tertiary structure of a transfer RNA (tRNA). The tRNA in Figure 40 and its structures in Figures 41a and 41b is a $tRNA^{Phe}$, thus encoding for the amino acid phenylalanine. A more detailed description of tRNA genes is given in Section 5.3.2.

Evolution of molecules can affect mutations in sequence while retaining the functional structure, thus one wants to compare sequences as well as corresponding regions of the secondary structure. Approaches to this problem use combinations of alignment and folding, DP algorithms based on the *Sankoff algorithm* (Sankoff, 1972).

As described in Subsection 2.3.2, RNA structure prediction is a major task in bioinformatics and there exist several programs calculating putative secondary structures for a given sequence. The first algorithm of this kind is the Nussinov algorithm, a DP algorithm described in Subsection 2.3.2. Until now, various energy parameter sets and structure prediction algorithms exist (Lorenz et al., 2016), trying to predict structures based on sequence data. Further methods include experimental data or try to extend the predictions to 3-D structures (Lorenz et al., 2016; Thiel et al., 2017). Extended structure prediction algorithms are based on various parameters such as energy, structure or environmental constraints. The `ViennaRNApackage` includes several programs to fold and visualize RNA structures based on different parameter settings. The protein data bank (PDB) (Berman et al., 2000) is a database listing tertiary structures of RNAs and proteins that have been experimentally verified.

5.4.4 Homology Search

Homologous sequences are assumed to have a high sequence and/or structure similarity. Statistical models such as hidden Markov models (HMMs) and covariance models (CMs) are frequently used to infer similarities and conservation of sequences and structures. They

use DP algorithms such as the forward-backward or Viterbi algorithm in order to estimate probabilities or find the most probable path through the search space (Durbin et al., 1998) (see Chapter 2). CMs are used to build models of RNA families based on MSA and predicted consensus structure. The program suite `Infernal` (Nawrocki and Eddy, 2013) includes programs to create CMs from a MSA (`cmbuild`) and further programs to scan a genome for sequences homologous to the model (`cmscan` or `cmsearch`). The tool `tRNAscan-SE` is specialized to detect tRNA sequences in genomes, based on various CMs of tRNAs (Lowe and Eddy, 1997). If `Infernal` is used to retrieve a set of target elements, the user can specify a threshold for the `Infernal` score that will mark an element as a pseudogene. In case of tRNA detection, `tRNAscan-SE` is used to retrieve a set of target tRNAs and is trained to detect tRNA types and pseudogenes.
The search for homologous protein sequences can be done using the tool `HMMer` (Eddy, 1998; Eddy, 2011). Here, models include consensus secondary structures as well as amino acid sequences. Thus, detecting homologous proteins is done using HMMs which can be created based on a MSA (`hmmbuild`) and used to scan genomes (`hmmscan`), analogous to its counterpart for non-coding RNA (ncRNA) sequences. Further detailed explanations on the basics are given in Durbin et al. (1998). HMMs and CMs were used in Chapter 6, Chapter 7 and Chapter 8 as a useful tool to analyze biological data and find homologous counterparts.

5.4.5 Reconstructing Phylogenetic History

A large arsenal of computational methods has become available to determine orthology that can be separated into three groups: (i) sequence based tools, (ii) tree based tools and (iii) graph or cluster based tools (Lechner et al., 2014). These methods can be separated again into character based or distance based methods. Most algorithms first require scored pairwise or multiple sequence alignments. Hereby, scores are either based on direct distances between sequences or character substitution models, assuming certain probabilities for the exchange (mutation) of single characters (nucleotides or amino acids).

Character based methods try to infer the optimal tree topology based on an optimality search criterion, e.g. maximum likelihood (ML) such as `RAxML` (Stamatakis, 2006), maximum parsimony (MP) such as `Paup*` (Swofford, 2004) or Bayesian methods, e.g. `MRBAYES` (Huelsenbeck and Ronquist, 2001).

A *distance based method* using an optimality criterion is the Fitch-Margoliash algorithm (Fitch and Margoliash, 1967; Fitch, 1970). This DP algorithm uses the MP approach to find a tree topology that includes all input sequences in a way that the number of mutations is minimized. A description of the Fitch algorithm based on algebraic dynamic programming (ADP) is given in Chapter 3 and the application of a modified version is described in Chapter 7.

Sequence based methods rely solely on sequence comparisons and use a "reciprocal best match" rule (Tatusov et al., 1997; Lechner et al., 2011). Here, it is assumed that orthologous sequences are more similar than paralogous or non-homologous sequences. There exist approaches including function annotation of genes to confirm composition of groups of orthologous genes which will be further described in Chapter 7 (Tatusov et al., 1997; Koonin, 2005; Galperin et al., 2014).

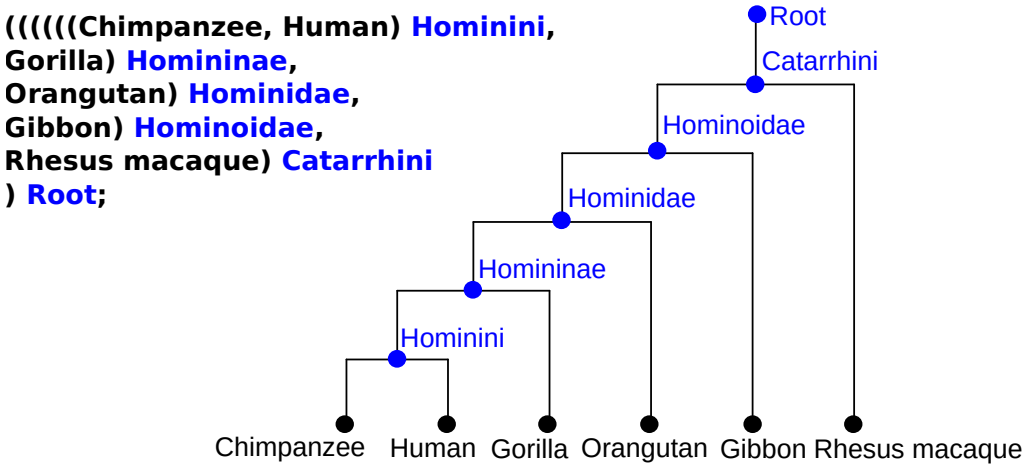


Figure 42: Phylogenetic tree of the order of *Catarrhini* (Old World Monkeys) with a selected set of species at the leaves of the tree (black nodes). Inner nodes (blue) show subgroups. The line on the top left displays the same tree in Newick format whereas labels of inner nodes are written in blue and leaf labels in black. Sibling nodes are written inside parentheses with their parent directly following the closing parenthesis starting from the most inner subtree on the left going towards the root.

Tree reconstruction based on pairwise distances between sequences uses hierarchical clustering to infer a best fitting tree (e.g. neighbor-joining (NJ) (Saitou and Nei, 1987)).

A further approach is to create *orthology graphs* where nodes represent genes and edges represent orthology relations between pairs of genes. It has been shown that the graph has to have a cograph structure in order to represent a true orthology relation (Hellmuth et al., 2013). Based on (reciprocal) best matches of sequences, it is possible to infer distance metrics when comparing sequences and create clusters of orthologous genes (Geiß et al., 2019).

Various extending models and heuristics exist such that the user has the choice between many possibilities on how to reconstruct a phylogenetic tree based on a set of sequences (Lemey et al., 2009). Reviews of the topic and benchmarks of the most commonly used tools can be found in Altenhoff and Dessimoz (2009), Kristensen et al. (2011), Salichos and Rokas (2011), Dalquen et al. (2013), and Ward and Moreno-Hagelsieb (2014). Most approaches assume that genes evolve essentially independently so that sequence divergence is a faithful measure of evolutionary distance.

There exist specified models for certain evolutionary processes describing the evolution of genes. In case of frequent unequal crossing over it is possible to detect genomic rearrangements using circular-split systems (Prohaska et al., 2018). In case of concerted evolution, syntenic information based on orthologous anchors in the genome can be taken as additional references to infer orthologous relations (Velandia-Huerto et al., 2016; Berkemer et al., 2017a) (see Chapter 6).

Including co-orthologous and co-paralogous genes, the orthology relation is a non-

transitive relation which makes sequence based methods error prone to overestimating orthologous relations and heavily rely on already annotated data. Tree based methods are more accurate, however, they are computationally expensive and not applicable for large sequence data. Graph and cluster based methods are less resource-intensive than tree-based methods, however, they start with estimating orthology relations and then edit the set of relations in order to retain consistent sets of mutually orthologous genes. Methods are described in more detail in Nichio et al. (2017), Kristensen et al. (2011), Koonin (2005), and Lechner et al. (2014).

A popular format to encode phylogenetic trees is called *Newick format* (Felsenstein, 1999). As displayed in Figure 42, it maps the tree structure to a string containing node identifiers separated by parentheses to distinguish subtrees. The root of the tree is usually displayed on the r.h.s. of the Newick string. A node's children are written in parentheses on the l.h.s. of the parent node separated by ','. This results in an expression of nested parentheses. In a Newick tree, it is also possible to encode branch lengths of the tree by adding the length behind the corresponding node identifier showing the distance to its parent, e.g. *Gorilla:0.5*.

CHAPTER 6

Duplication Alignments to Reconstruct Evolutionary History

Contents

Repetitive elements evolved due to a high number of duplication events in the genome as described in Section 5.3.2. Hereby, functional genes only show a low number of mutations and are highly similar. This process is called concerted evolution. In order to trace back the evolutionary history for these genes, it is not possible to only look at sequence similarity but also include information based on the order of genes, called synteny.

This chapter is mainly based on Berkemer et al. (2017a) titled *SMORE: Synteny Modulator Of Repetitive Elements* and on Velandia-Huerto et al. (2016) titled *Orthologs, turn-over, and remolding of tRNAs in primates and fruit flies*. It describes a systematic conceptual workflow for the evolutionary analysis of multicopy genes. As an input, it uses genome-wide multiple sequence alignments (MSAs), many of which are already publicly available, as source of synteny information. The underlying theory is developed in Velandia-Huerto et al. (2016) and a fully automated pipeline that serves as a convenient tool for this purpose together with applications to two classes of ncRNAs, namely tRNAs and Y RNAs is given in Berkemer et al. (2017a). A central part of the workflow and pipeline is a dynamic programming (DP) algorithm used to compute optimal solutions for duplication alignments, hence, alignments that allow matches of the form 1:n or n:1, respectively. The underlying theory is described in Section 6.3. Further information and underlying data can be found in the supplemental files of Velandia-Huerto et al. (2016) and Berkemer et al. (2017a).

6.1 Concerted Evolution

Concerted evolution (Liao, 1999; Nei and Rooney, 2005) may cause paralogous genes to maintain essentially identical sequences over long evolutionary time scales. The underlying mechanism is primarily homologous recombination, which leads to gene conversion where a piece of sequence from one copy of the gene effectively overwrites a homologous region in another copy. Unequal crossover between repeating units and gene amplification are also important contributors, see e.g. Liao et al. (1997). Gene conversion is responsible for preventing the divergence of the individual copies of tRNA (Amstutz et al., 1985), snRNAs (Liao et al., 1997), the ribosomal RNA cistron (Naidoo et al., 2013), and the histone genes (Scienski et al., 2015). Paralogous genes can escape from concerted evolution (Teshima and Innan, 2004) and then rapidly accumulate mutations typically leading to loss of function and hence eradication from the genomic record. Together, these processes can result in a rapid net turn-over of gene copies and sometimes large differences in the number of copies in closely related genomes. This effect has been studied in much detail in particular in the case of tRNAs (Bermúdez-Santana et al., 2010; Rogers et al., 2010; P. P. Wang and Ruvinsky, 2012; Rogers and Griffiths-Jones, 2014).

Since paralogous sequences are essentially identical, it is not possible to identify orthologs of genetic elements that are subject to concerted evolution by means of sequence comparison. Synteny, however, provides a potentially powerful means of discriminating orthologous loci. Reliable information of synteny can be obtained whenever there are unique sequence regions in close genomic proximity of the locus of interest. Here, orthology can be established with high confidence among related species. The conservation of proximity to such independently evolving regions can then be used to distinguish orthologous from paralogous copies of the ambiguous sequence element. This idea has been exploited in the

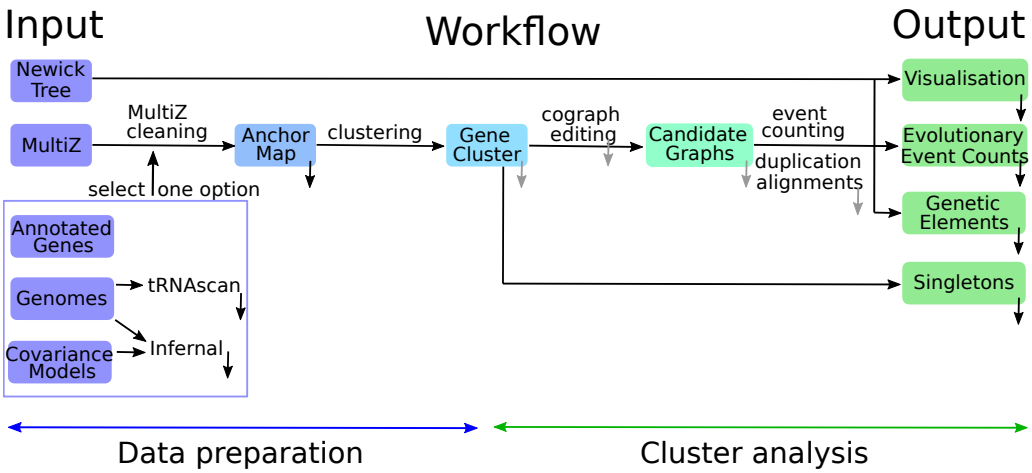


Figure 43: Summary of the computational workflow implemented in the `SMORE` pipeline for analyzing the evolution of multicopy genes. The compilation of orthology estimates (data preparation) and the quantitative analysis (cluster analysis) are logically separated and can also be used independently of each other. See text for details. The blue box describes options for input data. Black arrows pointing towards the next step of the pipeline (to the right) show an uninterrupted workflow. Black arrows pointing downwards indicate output files that are always part of the output whereas gray arrows pointing downwards indicate the creation of temporary files and of optional output for the user, respectively.

past in particular as means of tracing the evolution of tRNAs (Bermúdez-Santana et al., 2010; Rogers et al., 2010; P. P. Wang and Ruvinsky, 2012; Rogers and Griffiths-Jones, 2014).

In this chapter, creation of gene clusters for repetitive elements is described to explore the concepts of concerted evolution at the example of two classes of ncRNAs. Multiple identical copies, often large numbers of pseudogenes, and rapid, lineage-specific expansions of particular families are typical for **tRNA** evolution at least in eukarya (Frenkel et al., 2004; Bermúdez-Santana et al., 2010). Among the elements under concerted evolution, tRNA genes are the best studied elements. They show a rapid turnover as the consequence of frequent seeding of new loci compensated by high rates of pseudogenization (Bermúdez-Santana et al., 2010; Rogers et al., 2010; P. P. Wang and Ruvinsky, 2012; Rogers and Griffiths-Jones, 2014). While gain and loss events can be estimated from changes in the total number of paralogs with often acceptable precision for low-copy-number gene families such as microRNAs (Hertel and Stadler, 2015), this is not the case for tRNAs since the number of conserved tRNA loci very quickly decreases with phylogenetic distance (Bermúdez-Santana et al., 2010; Velandia-Huerto et al., 2016).

The evolution of **Y RNAs** has been studied in some detail in Mosig et al. (2007), indicating a single, evolutionary conserved genomic cluster comprising four paralog groups designated Y1, Y3, Y4, and Y5. With the notable exception of mammals, which harbor on the order 1,000 Y RNA derived retro-pseudogene sequences (Perreault et al., 2005),

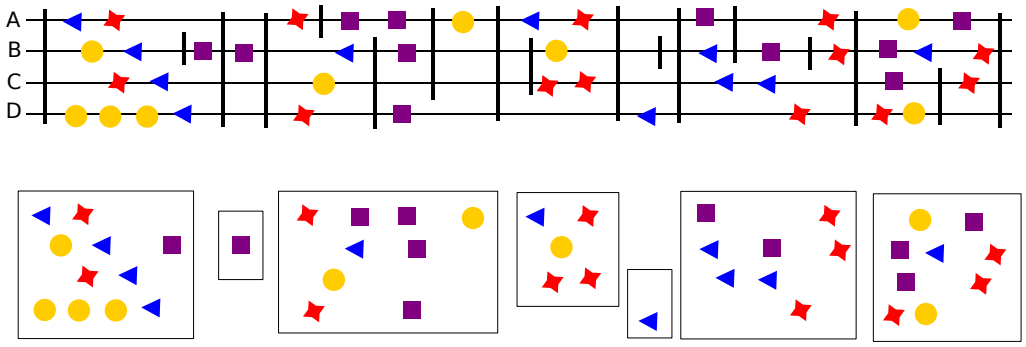


Figure 44: Example of syntenic gene clusters before post-processing. Top: Horizontal lines A, B, C and D show genomic regions of four distinct species. Black vertical lines show genomic anchors that possibly have orthologous counterparts in other genomes. They represent the syntenic anchors and define the borders of gene clusters. Distinct genetic elements are depicted by colored shapes. Bottom: clusters are separately shown as framed entities.

most other vertebrates show only a few Y RNA derived pseudogenes. Homologs of Y RNA in nematodes are called sbRNAs (stem-bulge RNAs) (Boria et al., 2010).

The pipeline is composed of two modular parts: (i) the inference of the orthology relation from the data and (ii) the quantitative analysis of the orthology relation and thus, evolutionary events, see Figure 43. The first component identifies a map of genomic anchor points that are used to partition the annotated elements of interest into an initial set of candidate clusters. These are then processed to account for the most common artefacts in the input data and refined using information that is provided by analyzing related but distinguishable sequence elements together. The second part of the pipeline is largely independent of the first and can also be employed using input data generated by other, third-party methods. With our pipeline, we provide an uninterrupted workflow that returns results based on input files and user-defined parameters. With the exception of breaks between subcommands indicated in Figure 43 and where output data is provided for the user, UNIX pipes are utilized to transfer data between software components.

6.2 Creation of Gene Clusters

The basic assumption for the creation of gene clusters is the existence of syntenic anchors within related genomes. These anchors are (mostly) protein coding genes for which there exists an orthologous counterpart in genomes of related species. Here, orthologous genes are identified based on sequence similarity (see Subsection 5.3.3 for further details).

Given a pair of genes g_A and g_B in species X and orthologous counterparts g'_A and g'_B in species Y, we can say that the genomic regions r_X and r_Y between the protein pairs in X and Y are orthologous. Thus, the syntenic order of orthologous proteins in distinct genomes defines orthologous genomic regions. Genes inside these regions are summarized to gene clusters which are then used to refine assignment of orthologous sets of genes.

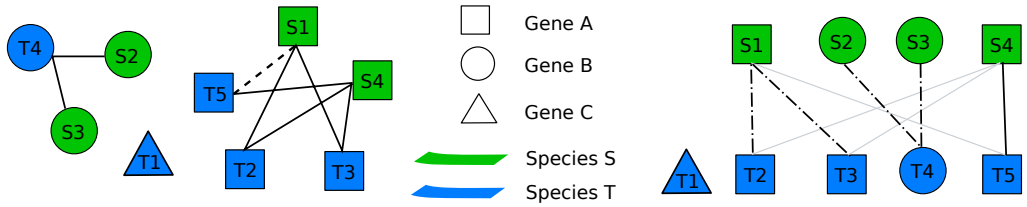


Figure 45: Right: Example of the graph G for a cluster consisting of two groups of orthologous elements in two species S (green) and T (blue). Thick edges indicate above-threshold sequence similarity. The dashed edge, which was included initially must be inserted to correct G: otherwise T5-S4-T3-S1 would form a P_4. Modified Needleman-Wunsch alignment for graph G (Left). The inserted edge to correct for a cograph is now part of the thick edges showing the orthology relation. The alignment will remove crossing edges of the orthology graph (grey) and detect duplications (dashed edges). Node T_1 is not aligned to any node in S which indicates a deletion in species S.

With this assumption, one can create anchor maps based on genomic multiple sequence alignments (MSAs) for several species. As not every protein has an orthologous counterpart in all species included in the MSA, a reference species is chosen to define a basic set of genomic anchors. Additionally, gene clusters can be divided or joined in between a fixed set of genomic anchors. A schematic example for the definition of gene clusters is shown in Figure 44. The creation of genomic anchor maps is explained in detail in A. Hoffmann (2020).

6.3 Quantitative Analysis of Evolutionary Events

The resulting partition of genes, thus, the gene clusters resulting from the anchor map may still contain non-orthologous elements. In the case of tRNAs, for instance the annotation generated by `tRNAscan-SE` only distinguishes anti-codon classes. These still may comprise multiple, discernible families. We therefore construct, for each cluster, a graph $G = (V, E)$ whose vertices are the annotated elements that belong to the cluster. An edge is drawn between two elements v and w if their sequences are more similar than a certain threshold. In the case of tRNAs, values of 80% to 90% sequence identity have proved useful (Velandia-Huerto et al., 2016). This value needs to be set specifically dependent on the typical sequence conservation of the elements under consideration and the phylogenetic range of interest. The graph G represents the orthology relation within a given cluster, see Figure 45 for an example.

As shown in Hellmuth et al. (2013) the graph G should be a cograph, i.e., it must not include a path P_4 on four vertices as an induced subgraph (for a definition, see Subsection 2.2.1). As G is constructed from the sequence data using fixed thresholds for sequence similarity, it will sometimes violate the cograph property. Nevertheless, it provides a good approximation. The initial graph G can be corrected by inserting or deleting a minimal number of edges that is required to restore the cograph property. Although cograph editing is known to be a difficult problem (the corresponding decision problem is NP-hard (Y. Liu et al., 2012)) it remains tractable for sizes of candidate graphs

we typically encounter.

The possibly edited graph G' may still overpredict orthology in cases where a cluster contains multiple types of elements that are distinguished by similarity. In such cases the order relative to dissimilar elements may subdivide the ortholog clusters of G'. To utilize this order information we consider an alignment of the element that (i) preserves their genomic order and (ii) allows matches only between elements that are connected by edges in G'. This variation of the alignment problem is solved by a variation on the well-known Needleman-Wunsch alignment algorithm (Needleman and Wunsch, 1970) that also allows duplications of elements, see Figure 45 for an example. As explained in Figure 45, the modified Needleman-Wunsch algorithm removes crossing edges and allows duplications. The exclusion of crossing edges is an intrinsic property of alignments and the reason for choosing this type of approach here. More precisely, alignment algorithms compute maximum weight matchings that preserve the prescribed order in both sets, when presented with two linearly ordered sets of objects (see also Chapter 4) and a weighted bipartite graph of allowed matches of pairs of objects from different sets. The modified version of the Needleman-Wunsch algorithm employed here extends the match case in such a way that one element in one set may also be matched with one or more consecutive objects in the other set.

6.3.1 Filtering Candidates with a Generalized List Alignment

The edited graph G' as described in the previous section is not sufficient to completely solve the orthology problem because we have no guarantee that any two tRNAs are separated by anchors. The available anchors in fact may enclose entire tRNA clusters, see Fig. 46. For tRNAs, however, we can clearly distinguish subgroups by sequence similarity. In particular, tRNAs of different isoacceptor families (i.e., those that are loaded with different aminoacids) and within these, most subgroups with distinct anticodons, exhibit clearly separate sequences. Members of these subgroups may serve as additional synteny anchors. However, they cannot be employed directly, because we only know that they preserve relative order, but due to their multi-copy nature, we cannot identify unique correspondences from sequence information alone.

Again, we rely on the approximation of strict order preservation. In the context of tRNA clusters, this amounts to the assumption that tRNAs within a gene cluster proliferate by means of single gene tandem duplications or by retroposition-like insertions. There might also be tandem duplications of subclusters. Well-established methods for resolving such scenarios (Elemento and Gascuel, 2005) are not applicable, because the tRNA genes of the same family have nearly identical sequences as a consequence of their concerted evolution, thus gene trees cannot be reconstructed.

The relationship between clustered tRNAs in two species corresponds to a generalized version of an alignment problem. To see this, we consider each tRNA cluster as an ordered list of tRNAs and tRNA pseudogenes t_i^a and t_j^b in the two genomes a and b. For the sake of the argument let us first neglect gene duplications and consider insertion, deletion, and remolding only. In this case the correspondences between tRNAs in the two clusters must be an order-preserving matching, i.e., an alignment, between the ordered sets $\{t_i^a\}$ and $\{t_j^b\}$. There are stringent restrictions on which tRNAs can actually be matched. In essence we have to require that the genetic distance $d_G(t_i^a, t_j^b) < \varepsilon$, where the threshold ε

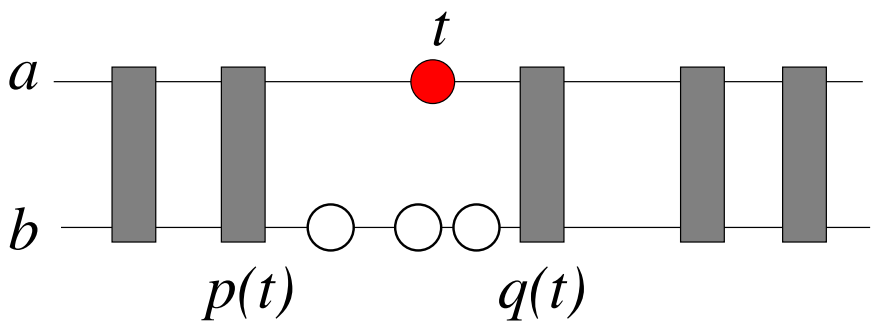


Figure 46: Tight anchors for t into species b. The tight anchors are the anchors closest to t that connect species a and b. By synteny, the only possible orthologs of t are the three loci indicated by white circles.

is chosen as an upper bound on divergence of genes in phylogenetic range of interest. If $d_G(t_i^a, t_j^b) \geq \varepsilon$ no match of t_i^a and t_j^b is allowed. This amounts to removing edges from G' that connect tRNAs for which we know that they cannot be orthologs based on sequence similarity.

In addition to regular insertions, deletions and $1:1$ (mis)matches, we also allow $1:q$ and $q:1$ mismatches to accommodate duplications. Since the scoring in list alignments is dominated by excluding significantly different items from matching at all, we settled for a simple scoring model of the form

$$\lambda(t_i^a, t_{j-1}^b, t_j^b) = \delta(t_i^a, t_{j-1}^b) + \delta(t_i^a, t_j^b) + \eta \tag{6.1}$$

for $1:2$ matches, where $\eta > 0$ is an extra penalty for the duplication and the two copies are otherwise scored independently like substitutions. $1:q$ matches for $q > 2$ are treated analogously. For two tRNAs we use the dissimilarity score $\delta(t', t'') = 20$, if Hamming distance between t' and t'' is below a threshold value, and $\delta(t', t'') = \infty$ for more different tRNAs. Whereas 20 is a fixed value for the first scoring, later $\frac{\delta(t',t'')}{length(tRNA)} = 0.9$ was used such that the differences between two tRNAs are 10% of their length. Since tRNA sequences have similar length overall the results are robust against such changes in the scoring function.

In order to account for local, i.e., order-preserving duplications we simply have to extend the alignment model: in the simplest case, we only allow 1:2 and 2:1 matches. This leads to the following simple modification of the Needleman-Wunsch (Needleman and Wunsch, 1970) algorithm:

$$D_{ij} = \min \begin{cases} D_{i-1,j-1} + \delta(t_i^a, t_j^b) & \text{M} \\ D_{i-1,j} + \delta(t_i^a, -) & \text{I}(a) \\ D_{i,j-1} + \delta(-, t_j^b) & \text{I}(b) \\ D_{i-1,j-2} + \lambda(t_i^a; t_{j-1}^b, t_j^b) & \text{D}(a) \\ D_{i-2,j-1} + \lambda(t_{i-1}^a, t_i^a; t_j^b) & \text{D}(b) \end{cases} \tag{6.2}$$

with the usual initialization score $D_{00} = 0$ for the empty alignment; The values $D_{i0} = \sum_{i' \leq i} \delta(t^a_{i'}, -)$ and $D_{0j} = \sum_{j' \leq i} \delta(-, t^b_{j'})$ correspond to the insertion of prefixes. Here M, I(a), I(b), D(a), and D(b) refer to (mis)matches between a and b, insertions in a and b, and duplications in a and b, respectively. An analogous algorithm can be devised by more complex $p : q$ duplication operations. Several variants of the simple Needleman-Wunsch alignment scheme to accommodate $p : q$ matches beyond simple $1 : 1$ correspondences have been proposed and applied to natural language data, see e.g. (Kondrak, 2000; Eger, 2013). A variant of list alignment has also been used in (Fried et al., 2004) to extract co-linear clusters of phylogenetic footprints.

The alignment edges predicted by the pairwise generalized list alignments serve our best estimates for the orthology relation. For 1:2 duplications, an edge is inserted from the "original" to both "copies"; in the more general case of p-to-q duplications, we accept all edges of the complete bipartite graph corresponding to the p-to-q duplication. Superimposing all pairwise alignments by construction again results in a subgraph of G'. We call this graph G_o. It contains only edges between tRNAs that can be orthologs according to their sequence similarity, and all connected components of G_o are order preserving since their edges result from the order-preserving alignment step, see Fig. 44. In general, it will consist of many small connected components, each consisting only of members of a single tRNA family that locally has expanded and contracted by duplication and loss events. An example of a more complex cluster is shown in Figure 47. Of the cographs, 327 were cliques and thus did not contain duplication events. The remaining 206 include duplication events that increased the total number of tRNAs by 66. In addition, 60 duplications were detected in the connected components containing only tRNAs of the same species.

6.3.2 Counting Evolutionary Events

Taken together, the construction of the orthology relation outlined above provides, for each final orthology graph, information on (i) the first appearance of the ortholog group, (ii) duplication events, and thus (iii) on the losses. This follows from the theory developed in Hellmuth et al. (2013) and Hellmuth et al. (2015) establishing the correspondence between orthology relations and event-labeled gene trees. Usually, one is primarily interested in placing duplication and loss events relative to a known gene phylogeny. Although it is not always possible to reconcile event labeled gene trees with species trees (Hernandez-Rosales et al., 2012), we found that our data are almost always "clean" enough to cause little problems in this respect because the final ortholog groups contain only very small number of locally occurring paralogs. We can therefore use a simple heuristic that corrects the graph structure by deleting or adding edges in such a way that they can be reconciled into a phylogeny. The heuristic iteratively deletes or adds edges in order to edit the structure. At the same time, the number of edges to be edited is kept minimal.

Given a species tree S and cluster C of orthologous genes, let $\sigma(x) \in S$ be the species in which element $x \in C$ resides. Thus $\sigma(C)$ is the set of species in which members of the cluster are attested. The appearance or *insertion* of C into S occurs within the edge ancestral to the least common ancestor ℓ of $\sigma(C)$ in S. As a consequence, every cluster that is present ancestrally is viewed as an "insertion before the root". Using the same parsimony assumption, we assume that deletions of C appear in the edge ancestral to

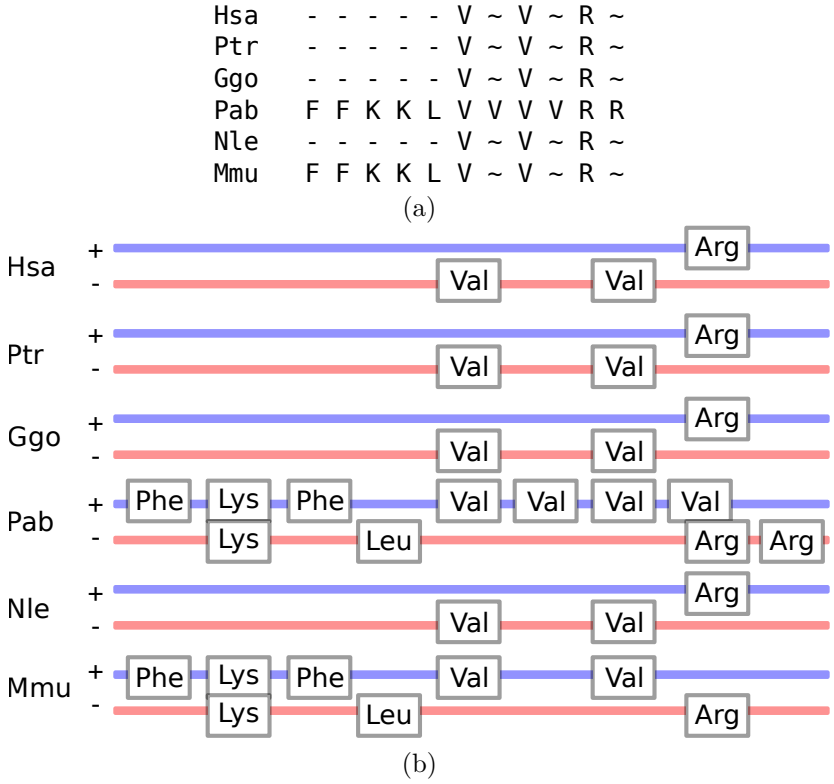


Figure 47: A more complex tRNA cluster in primates. Panel (a) summarizes the situation as list alignment. For simplicity, tRNAs from both strands are included. Gaps in the alignment are indicated by $-$, duplication events are shown by $\sim$. Panel (b) shows a more detailed, strand-specific genomic map. It highlights the reversal of the orientation of a tRNA Arg (R) in *Mmu* and the two copies of tRNA Lys (K) on opposite strands. The tRNAs isoacceptor classes are indicated by their 1-letter codes in panel (a): Phe (F), Lys (K), Leu (L), Val (V), Arg(R). Species abbreviations are *Hsa*: *Homo sapiens*, *Ptr*: *Pan troglodytes*, *Ggo*: *Gorilla gorilla*, *Pab*: *Pongo abelii*, *Nle*: *Nomascus leucogenys*, *Mmu*: *Macaca mulatta*. Common species names are listed in Table 3.

maximal subtrees S' of S below ℓ that do not contain species from $\sigma(C)$. If the species tree is fully resolved, then deletions are never inferred at an edge leading to a child of ℓ.

If a cluster contains multiple paralogs, duplication events are associated with changes in copy number. Since clusters are by construction local in the genome, such duplication events corresponds to tandem duplications. In contrast, the proliferation of the elements by insertion at different loci is accounted by the insertion events. A detailed mapping of tandem duplications to the species trees is non-trivial since the event-labeled gene trees obtained from cographs are usually not fully resolved. The pipeline therefore counts only the duplication events that occurred along the lineage leading from the root to a given leaf. This information can be extracted directly from the pairwise alignment of the element

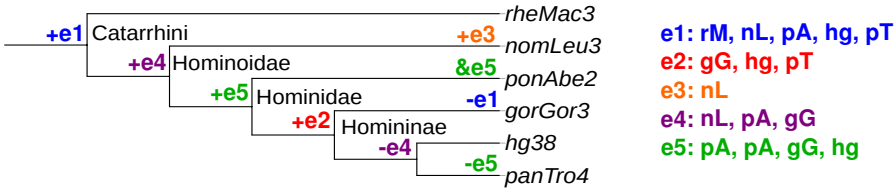


Figure 48: Example for counting genetic events. (e1) - (e5) are five groups of orthologous elements. '+' and '-' signs show where insertions and deletions are counted in the tree based on the groups. '&' depicts a duplication. Deletions can possibly be reported as missing data, too. rM, nL, pA, gG, hg and pT are abbreviations for species identifier *rheMac3, nomLeu3, ponAbe2, gorGor3, hg38* and *panTro4*. Species names and corresponding abbreviations are listed in Table 3.

orders within each cluster. An example is shown in Figure 48.

6.4 Results

This chapter describes a fully automatized pipeline that implements an improved version of the conceptual workflow of Velandia-Huerto et al. (2016) for the detailed quantitative analysis of genetic elements that are subject to concerted evolution. It uses synteny information provided by uniquely aligned sequences adjacent to the multi-copy elements of interest as the key information to disentangle their evolutionary relationships. The mathematical properties of orthology relations and their equivalent to event-labeled gene trees guides the post-processing of the data. This makes it possible to obtain an accurate and very well resolved picture of the history of multi-copy families. The pipeline is publicly available and not only greatly facilitates the analysis in practise but also ensures a high degree of reproducibility. For convenience, the pipeline also includes options to automatically generate input annotation data using `tRNAscan-SE` and `Infernal`. The output of the pipeline includes files to easily visualize the resulting phylogenetic tree using `iTOL` (Letunic and Bork, 2016), thus facilitating the interpretation of the results.

The functionalities of the pipeline were tested with two sets of repetitive elements and data sets for six and ten mammalian species. tRNAs and Y RNAs were detected using `tRNAscan-SE` and `Infernal` and anchor maps were created for each data set and gene family. The data set with six species consists of six primates including human. The larger data set with ten species is an extension thereof and additionally includes the outgroups dog and mouse. Table 3 gives a summary of species included in the reconstruction of tRNA and Y RNA phylogenetics.

6.4.1 tRNAs

The comparison of the six species and the ten species data shows an interesting effect: lineage specific deletions of tRNAs seem to be very frequent in mammals, see Figure 49. Including three additional outgroups substantially increases the number of tRNA loci that predate the ancestor of the Catarrhini. While in the six species data set 206 of the 731 human tRNAs are placed at the ancestral branch, the number increases to 328 in the ten species set. This is compensated by a correspondingly larger number of lineage-specific

abbreviation	species	common name
canFam3	*Canis lupus familiaris*	Dog
gorGor3	*Gorilla gorilla gorilla*	Gorilla
hg38	*Homo sapiens*	Human
rheMac3	*Macaca mulatta*	Rhesus macaque
mm10	*Mus musculus*	Mouse
nomLeu3	*Nomascus leucogenys*	Gibbon
panTro4	*Pan troglodytes*	Chimpanzee
papAnu2	*Papio anubis*	Baboon
ponAbe2	*Pongo abelii*	Orangutan

Table 3: List of species and corresponding abbreviations used in the study.

losses in the outgroup species and a reduction of predicted insertion events in the human lineage.

Remolding of tRNAs was analyzed for the ten mammalian data. Although the exact numbers depend on the choice of the similarity threshold and the details of the cluster-joining procedure, we recovered most of the remolding events previously described in Velandia-Huerto et al. (2016) and Rogers and Griffiths-Jones (2014). As in previous reports, the overwhelming majority of remolding events concern pseudogenes and/or are lineage specific and most likely are the first steps in tRNA pseudogenization.

6.4.2 Y RNAs

Our data suggest that the spread of Y RNA sequences is an on-going process in mammals. Of the 990 loci identified, 190 date back to the ancestor of the Catarrhini, while on the order of a hundred loci have been inserted in both human and the chimpanzee lineage after their divergence, see Figure 50. The six and ten species data sets are largely consistent, although the inclusion of an additional member of the Cercopithecinae places many insertions that are estimated to be specific to Hominoidae and Catarrhini. Only a very moderate number of Y RNA loci was populated already in the ancestor of Simiiformes.

The copy numbers of the Y RNA families are comparable with the data reported in Perreault et al. (2007). Within Catarrhini there are consistently more Y1 and Y3 genes than Y4 loci. The number of Y5 copies remains small throughout the clade. Consistent with Perreault et al. (2007), our data show an appreciably level of syntenic conservation of Y RNA loci also beyond the Y RNA cluster that typical harbors on functional copies of each of the four families (Mosig et al., 2007).

6.5 Benchmarking and Application to Artificial Data

In order to test the functionality and performance of the pipeline, we constructed artificial data sets comprising six species with artificial “genomes” that are initially linked by 10.000 genetic anchors. 100 simulated “genetic elements” subdivided into three distinct types were randomly placed between the anchors. We considered both a random placement of

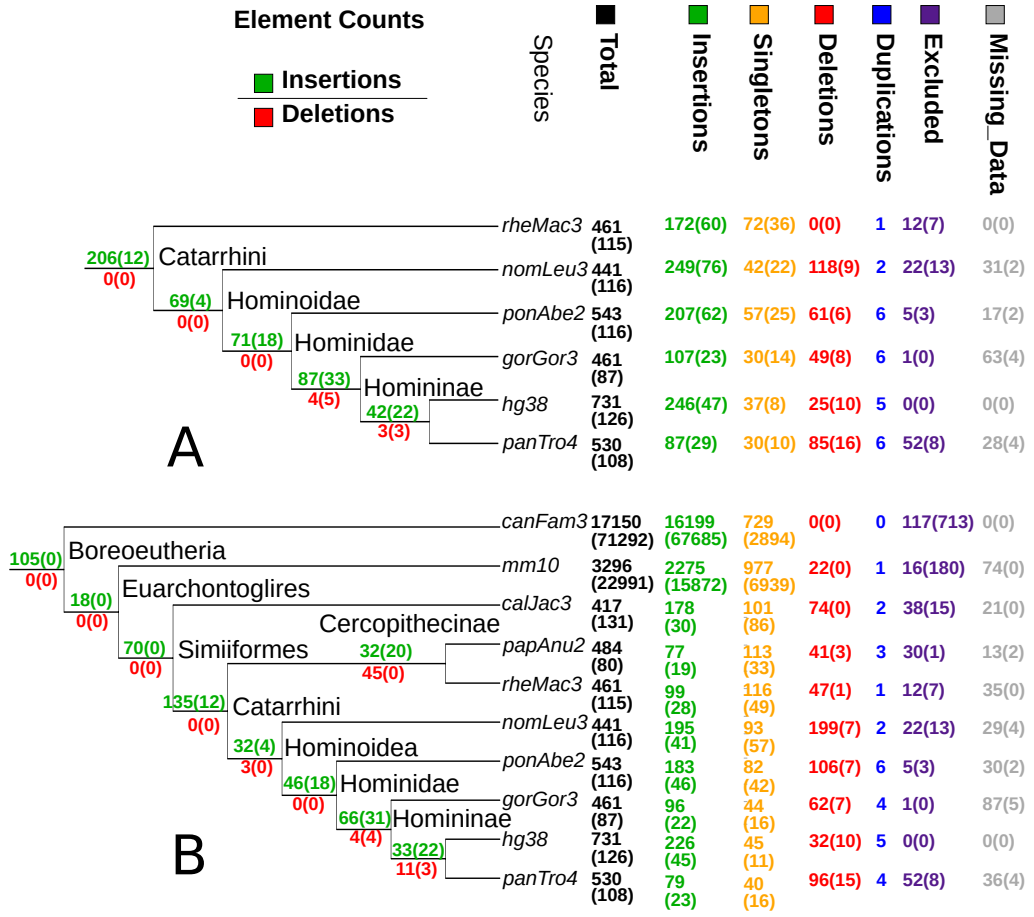


Figure 49: Summary of the evolutionary events inferred for tRNAs in an evaluation with six (A) and ten (B) species. Insertions and deletions that occur for groups of orthologous elements are inserted at their lowest common ancestor and possible deletions are added below the interior branches to which they refer. Other events such as singletons and duplications are added directly at the leaves for each species separately. Orthology relations are based on a similarity threshold of 80 % sequence similarity and clusters were joined using the relaxed adjacency constraints. Numbers in parentheses are numbers of pseudogenes. Full species names and corresponding abbreviations are listed in Table 3.

elements and the insertion of elements into homologous positions of all or a subset of the species. In order to model tandem duplication, furthermore, a fraction of elements was added twice. In order to simulate noise in the genome-wide alignments, a fraction of the anchor blocks was deleted randomly. We considered perfect data as well as a loss of 20% and 40% of the anchor blocks, respectively. For each setting, we executed our pipeline and compared the reconstructed orthology assignments and gain/loss statistics to the known ground truth.

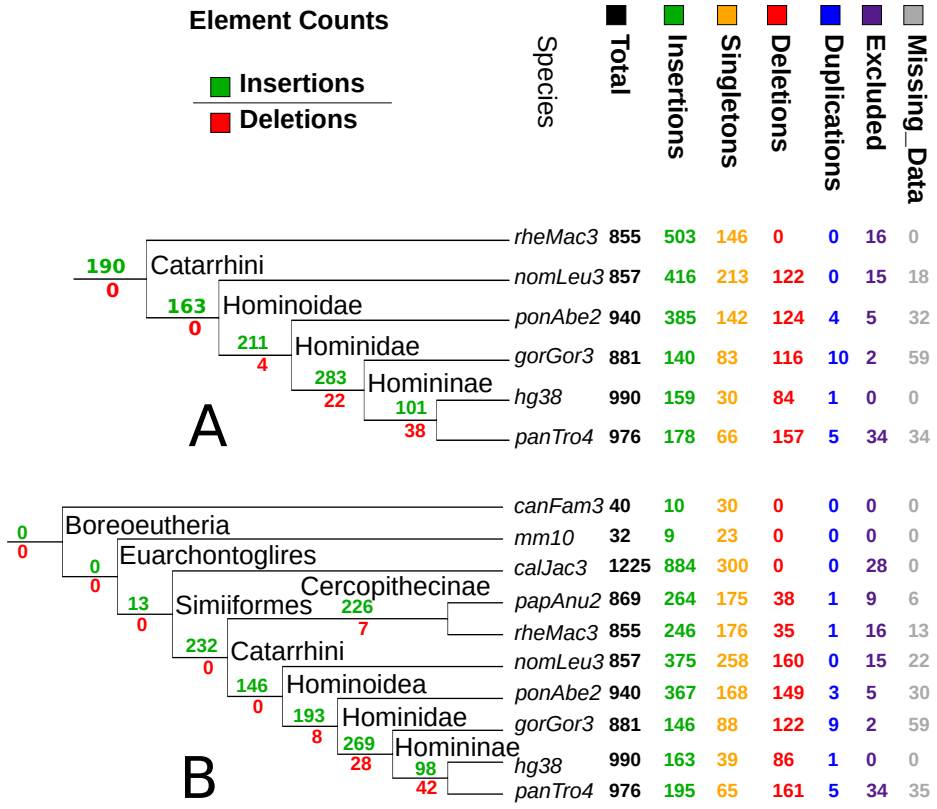


Figure 50: Summary of the evolutionary events inferred for Y RNAs in an evaluation with six (A) and ten (B) species. See the caption of Figure 49 for a detailed legend. The main difference between the two data sets is that the inclusion of an additional member of the Cercopithecinae moves a substantial number of the insertion events from Hominoidae to Catarrhini.

As described in above, artificial data sets were created using distinct levels of noise, hence perfect data, 20% loss and 40% loss of genomic anchors. Using perfect data, i.e., no deleted blocks, the pipeline exactly reconstructed the ortholog groups. With increasing noise level, the number of singletons decreased and the number of inferred local duplications increased since loci are joined upon loss of intervening anchors. With increasing noise level, an increasing fraction of deletion events is classified as missing data. At the same time, we observe an increase of inferred insertions at interior nodes of the tree, owing to a failure to correctly assign an ortholog from an outgroup. This is also depicted in Figure 51. Both effects are expected and cannot be addressed at the level of synteny data. In order to counteract this issue, more accurate and complete genome-wide alignments would be necessary.

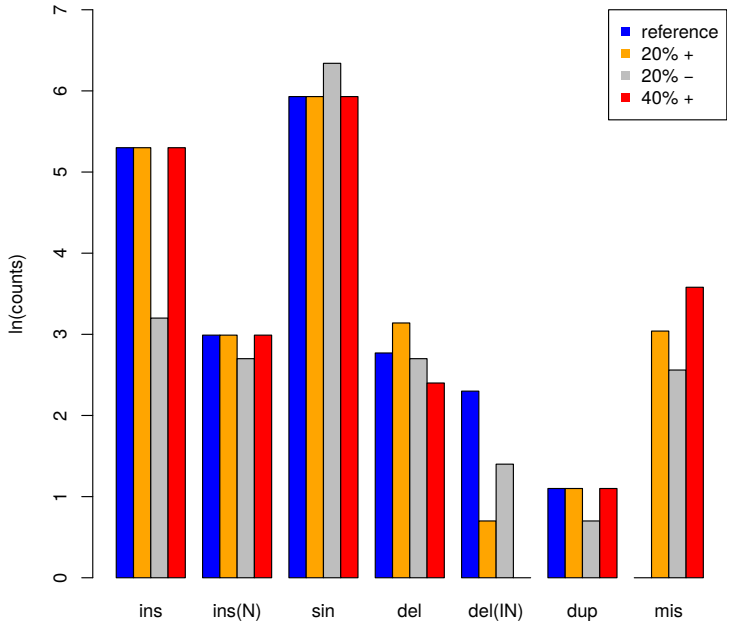


Figure 51: Summary of results for simulated data. The final counts (counted as natural logarithm ln(counts), y-axis) for evolutionary events, i.e., insertions (ins) and deletions (del) at the leaves, insertions (ins(IN)) and deletions (del(IN)) at the interior nodes, singletons (sin) and potentially missing data (mis) are compared between the reference ground truth and alignment with 20% (orange and grey) and 40% (red) of missing anchors. For 20% noise level we also compare the results with (+, orange) and without (-, grey) the segment joining step. High levels of noise mostly lead to a reduction in the inferred number of deletion and a corresponding increase in the reporting of missing data. Employing the joining strategy in general yields much more accurate results. Omitting the joining step in particular leads to smaller numbers of insertions inferred for interior nodes.

6.6 Implementation

The pipeline, which is written in `Python` and `Perl`, is available from `https://github.com/AnneHoffmann/Smore`. It requires `Infernal` and `tRNAscan-SE` if the user decides to use these tools for the genome annotation step. A user manual provides detailed usage instructions. We additionally include a small example in the repository giving instructions on how to apply the pipeline to data. Input data and output files for all subcommands applied on the small test set are available, respectively. The repository also provides the covariance models and the gene lists used in this contribution. As show-case examples we investigated the evolution of several multi-copy ncRNAs families. First we reanalyzed the evolution of tRNAs in two different mammalian data sets, comprising six and ten species respectively. Then we consider the much less studied Y RNAs for mammals and nematodes.

Both parts of the pipeline run fully automatized based on the given input and parameters.

Hence, the second part is available in two different versions, a fast version with as few output files as possible and a slower, verbose version that will print intermediary files such that the user can have a deeper and more detailed look into the data. This includes the formation of clusters and graphs created thereof as well as derived duplication alignments used for counting phylogenetic events.

The current version of the pipeline requires the following input data:

i) A multiple sequence alignment of the genomes under consideration is required to extract the synteny anchor points. Currently only `Multiz` format is supported.

ii) The corresponding genomic sequences are required for the annotation of the loci of interest. The pipeline expects `fasta` format. Since there is no guarantee that genome-wide MSAs represent the complete genome, both MSA and genomes must be provided.

iii) Target elements can be specified either as user-supplied annotation files, or as one or more covariance models for annotation with `Infernal` or `tRNAscan-SE`. The modular organization of the pipeline makes it straightforward to add, in future releases, further means of generating annotation information, such as Hidden Markov Models of proteins.

iv) A phylogenetic tree of the species of interest is necessary as a background to which evolutionary events are mapped.

The first three data items are required for the construction of the orthology relation. The phylogenetic tree is required only for the second part of the pipeline.

There are several parameters that can be adjusted by the user. The most important one is the similarity threshold for true orthology candidates. For the show-case examples reported here we used the same threshold value of 80 %. The threshold for low scoring `MAF` blocks that are to be discarded from the analysis can also be determined by the user. In addition, the pipeline offers several command line parameters to only run on subsections of the workflow and to omit some of the intermediate processing steps. For details we refer to the user manual.

The pipeline produces both machine-readable text files containing details of the analysis and condensed representations. The pipeline can also store detailed information on intermediate results that may be useful in particular also as a starting point to explore alternative analysis strategies. The final results include (i) the main results file, a phylogenetic tree displaying the evolutionary events in `newick` format as well as auxiliary files for the visualization of the tree and event information using `iTOL` (Letunic and Bork, 2016), (ii) a file listing all gene clusters retrieved from the input data, (iii) a list of all genetic events sorted by event and species, (iv) a list containing the numbers of genetic elements sorted by species and type, and (v) a list containing remolding events. We also used `iTOL` (Letunic and Bork, 2016), an interactive online visualization tool to generate the results tree. Optional intermediate files include (i) the edge-weighted graph of each initial cluster, (ii) a file for each of the clusters specifying which elements are contained in the cluster including all available annotation information for each element, (iii) the element-wise alignments of each cluster, (iv) information on the cograph structure or deviations thereof.

6.7 Concluding Remarks

The methods of molecular phylogenetics require a strong correlation between sequence similarity and evolutionary divergence times. Since the mechanisms of concerted evolution obliterate this correlation, molecular phylogenetics is not applicable to the analysis of multi-copy gene families including tRNAs and many other ancient ncRNA families. This limitation can be overcome in a systematic manner by using synteny, that is, conservation of relative gene orders, to identify orthologous elements. The fully automated pipeline presented here faciliates such an analysis of gene families that evolved due to concerted evolution.

CHAPTER 7

Dynamic Programming on Phylogenetic Trees:
Towards the Last Common Ancestor

Contents

Comparative genomics and molecular phylogenetics are foundational for understanding biological evolution (C. R. Woese and Fox, 1977; Goldenfeld et al., 2017). Genes that are resistant to lateral gene transfer and non-orthologous replacement form a basis for historical inferences, and have been found to be mostly associated with transcription, translation and DNA replication. Phylogenetic domain separation is a classic way of inferring presence in the common ancestor (C. R. Woese, 1987; C. R. Woese et al., 1990), and conserved presence analyses indicate genetic traits of the last common ancestor of archaea and bacteria (Koonin, 2003; Harris et al., 2003; Mirkin et al., 2003; Charlebois and W. F. Doolittle, 2004; Becerra et al., 2007; Glansdorff et al., 2008). The processes of horizontal gene transfer, variable rates of gene loss and gain over time, and non-orthologous displacement, however, all blur the lines of vertical descent.

This chapter lists various approaches for the inference of genetic traits that might have existed in a common ancestor of archaea an bacteria. Phylogenetic trees of protein families reconstructed from an established reference set (Tatusov et al., 1997; Koonin, 2005; Galperin et al., 2014) are used to gain insights into the conflicting nature of literature reports which aimed to identify proteins in the lowest universal common ancestor (LUCA). One can calculate a comparative measure for phylogenetic trees using branch lengths and evolutionary distances between genes. There are a number of existing data sets which have given insights into the protein repertoire of LUCA (reviewed in Becerra et al. (2007)). The current chapter includes some examples for well-known groups of proteins such as ribosomal proteins, oxygen related genes and subunits of the CODH/ACS enzyme complex. A difference between small and larger ribosomal subunits can be observed whereas proteins related to oxygen metabolism and CODH/ACS enzymes are grouped together.

This chapter is based on Berkemer and McGlynn (2020) titled *Phylogenetic domain separation of protein families constrains functional inference of LUCA*. There exist various analyses based on groups of orthologous proteins in archaea and bacteria. The comparison of those data sets is the central part of this chapter and based on the reconstruction of phylogenetic trees from multiple sequence alignments (MSAs). A modified version of the Fitch algorithm (Fitch, 1971), a dynamic programming (DP) algorithm (see also Chapter 3), is used to calculate the minimum number of splits needed to separate archaeal and bacterial genes in the trees. This is combined with a measure for pairwise distances in the tree. Further information can be found in the supplemental tables in Appendix B and the supplemental files corresponding to Berkemer and McGlynn (2020). The programs used in this chapter are available online (`github.com/bsarah/treeSplits`).

7.1 Orthologous Proteins

Previous works that aimed at identifying protein families associated with LUCA, differ in methodology and conclusions. Harris et al. (2003) worked with fully sequenced genomes and used the conserved orthologous groups (COGs) (Tatusov et al., 1997; Koonin, 2005; Galperin et al., 2014) as a basis set. They found 80 COGs were conserved in presence in the genomes available at the time (Harris et al., 2003). 50 of these separated the archaea, bacteria, and eukaryotic domains upon phylogenetic analysis. A recent study in 2016 (Weiss et al., 2016) constructed clusters of orthologs de novo, and focused on protein families which phylogenetically separated the archaeal and bacterial taxa, in line

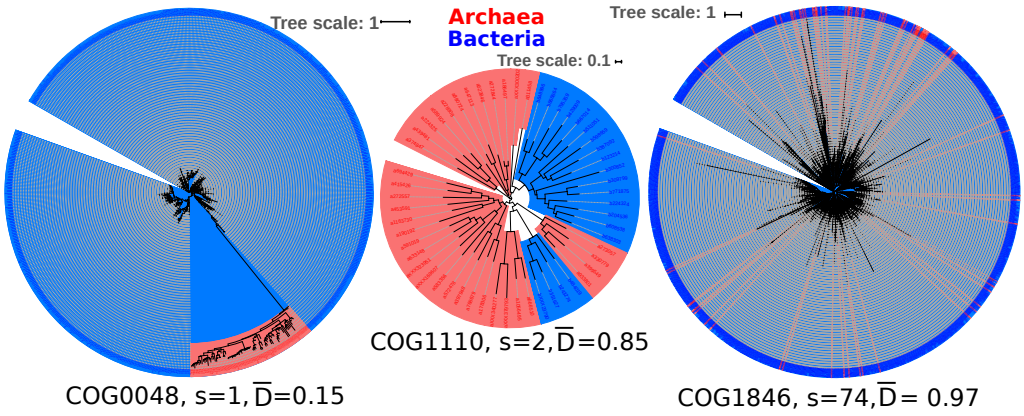


Figure 52: Reconstructed trees for COG0048 (Ribosomal protein S12), COG1110 (Reverse gyrase) and COG1846 (DNA-binding transcriptional regulator, MarR) with corresponding inter-domain splits (s) and $\overline{D}$ values, explained in Section 7.3.

with recent data that eukarya are derived from archaea (Raymann et al., 2015; Zaremba-Niedzwiedzka et al., 2017). There, 355 orthologous groups were reported to separate the two domains and inferred to be present in LUCA.

There are various methods to estimate orthology relations between genes as shortly described in Subsection 5.4.5. The set of conserved orthologous groups of proteins (COGs), developed and described in Tatusov et al. (1997) and Galperin et al. (2014), is created based on archaeal and bacterial proteins. The set of COGs consists of 4631 groups of archaeal and bacterial proteins where a second version additionally includes eukaryotic proteins (Galperin et al., 2014). The COGs are labeled by unique IDs as shown in Figure 52. Pairwise sequence alignments of those amino acid sequences are used to retrieve comparison scores and thus, best matching proteins between any pair of species. The proteins are then grouped together, even though not all proteins in a group are considered to be a best match. The creation of COGs additionally considers functional categories of the proteins as a measure such that proteins in the same group have similar functions. Data for each COG can be downloaded as multiple sequence alignments or the corresponding HMM which can be used to find further homologous proteins that might belong to the same group of proteins (`https://www.ncbi.nlm.nih.gov/COG/`).

The set of COGs are an established and known data set and the basis of various origins of life studies (Harris et al., 2003; Puigbò et al., 2009; Goldman et al., 2012; Charlebois and W. F. Doolittle, 2004; S. Liu et al., 2018; O'Malley and Koonin, 2011). However, the definition of orthology based on sequence comparison and function annotation might be subject for debate and is termed the *orthology conjecture* (Forslund et al., 2017). Especially protein annotation and corresponding functions is prone to various interpretations and errors (D. Lee et al., 2007). An approach to improve prediction of orthologs for sets of proteins could be to split proteins into their domains and only predict orthology of single protein domains. This accounts for recombination and losses within protein sequences, however, perfect detection of protein domains is not possible yet and even lacks a clear

Name	Total	Domain Sep.	Underlying Data Set
SSC	286514	355	cluster created by Weiss et al. (2016)
SSC^{COG}	335	44	SSC populated w. corresp. COG sequ.
Conserved COGs	80	50	COG, Harris et al. (2003)
mixed COGs	2886	665	COG, Tatusov et al. (1997)

Table 4: Table listing various data sets, their total number of gene groups, the number of domain separating groups of genes and how the groups were created. Number of domain separating groups are based on claims by Weiss et al. (2016) and Harris et al. (2003), whereas numbers for mixed COGs and SSC^{COG} are based on our own analyses as explained in the main text.

definition of a protein domain (Forslund et al., 2017).

In Weiss et al. (2016), protein clusters were created based on pairwise comparison of protein sequences. If the resulting score was higher than a previously set threshold, protein sequences were clustered together. The clusters are independent of function annotation, however, many of them have been assigned to a corresponding COG by the authors.

Catchpole and Forterre (2019) analyzed 3 groups of proteins and extended the corresponding COGs by homologous sequences from different organisms detected using HMMs. In this way they show for 3 well conserved groups of genes and one showing string indications for horizontal gene transfer which will be explained in more detail in subsequent sections.

Table 4 gives an overview over data sets mentioned in this chapter. The data sets consist of groups or clusters of orthologous archaeal and bacterial proteins. Corresponding phylogenetic trees reflect ancestral relations between archaeal and bacterial proteins. It is assumed that proteins that show a clear separation of sequences in archaea and bacteria are well conserved and might have existed in LUCA, see e.g. Weiss et al. (2016) and Harris et al. (2003).

7.2 Topology of Phylogenetic Trees

The data sets listed in Table 4 were used as a basis to reconstruct phylogenetic trees, which is described in more details in the following subsections.

7.2.1 Reconstructing Phylogenetic Trees

Our reconstruction of phylogenetic trees is based on three different data sets: COGs by Tatusov et al. (1997), conserved domain separating COGs by Harris et al. (2003) (which we call 'conserved COGs' in the following), and single split clusters (which we denote by SSC) created by Weiss et al. (2016). In their data sets, Harris et al. (2003) list 80 COGs with 50 having a single split topology and Weiss et al. (2016) discovered a set of 355 clusters that separated archaeal and bacterial species. We extracted the COGs that consist of bacterial as well as archaeal proteins and took the best matching proteins in case several paralogous sequences were found.

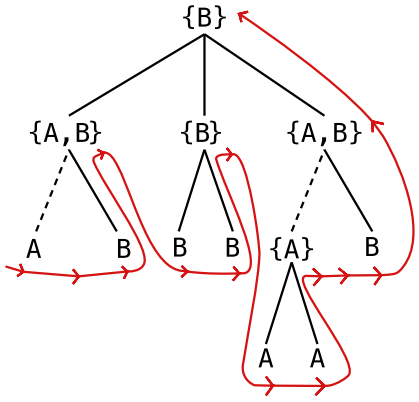


Figure 53: An example for the bottom-up phase of the modified Fitch algorithm. Leaves of the tree are labeled only by A and B. Inner nodes of the tree show sets of labels based on their leaves. Dashed lines indicate union operations during the bottom-up phase, thus this tree would need two splits to separate archaeal and bacterial genes. The red arrow indicates the postorder traversal of the tree.

In order to compare different approaches, we downloaded multiple sequence alignments (MSAs) for COGs (Tatusov et al., 1997) (`https://www.ncbi.nlm.nih.gov/COG/`) and collected corresponding COGs given in Harris et al. (2003) and Catchpole and Forterre (2019). Data supporting the conclusions reported in Weiss et al. (2016) were not included in the publication and were instead obtained from author contact on the now defunct pubmedcommons site (`ftp://ftp.ncbi.nlm.nih.gov/pubmed/pubmedcommons/`).

We used `FastTree` (Price et al., 2010) to reconstruct trees. The study of Weiss et al. (2016) used `RaxML` (Stamatakis, 2014) to build phylogenetic trees. However, we obtained almost the same results and thus, we used `FastTree` (Price et al., 2010) for all tree reconstructions. Thus, we obtain sets of trees for the set of COGs, SSC and conserved COGs.

7.2.2 Calculating Splits

A modified version of the Fitch algorithm (Fitch, 1971) was used to calculate the minimum number of splits s needed to separate archaeal and bacterial genes in the phylogenetic tree. The algorithm is a DP algorithm and its input is a single binary phylogenetic tree given in Newick format (see also Subsection 5.4.5). As described in Subsection 2.1.1, a memoization table is needed. Here, the memoization consists of a simple list which stores the labels at the nodes of the tree as depicted in Figure 53. Labels at the leaves of the tree correspond to species identifiers for archaeal and bacterial species. Labels at the inner nodes of the tree are sets of species identifiers from the leaves, thus the list is initialized with a neutral value, e.g. $\emptyset$ for the inner nodes and the species identifiers as labels of the leaves $\{label(l_i)\}$ for each leaf $l_i \in \mathcal{L}$. The bottom-up traversal of the tree is done based on the postorder, thus the children $c, d \in children(p)$ are visited before their parent p.

Then the label of p is determined in the following way:

$$label(p) = \begin{cases} label(c) \cap label(d) & \text{if } label(c) \cup label(d) \neq \emptyset, \\ label(c) \cup label(d) & \text{otherwise.} \end{cases} \tag{7.1}$$

The number of union operations is then the minimum number of 'mutations', thus a simple scoring would be to just sum up the union operations for the input tree. In order to know where the mutations happen, one needs to calculate the top-down phase, this time in preorder as the parent node p is visited before its child c, starting with the root of the tree.

$$label(c) = \begin{cases} label(p) & \text{if } label(p) \in label(c), \\ \text{arbitrary label from } label(c) & \text{otherwise.} \end{cases} \tag{7.2}$$

For this project, the set of labels of the phylogenetic tree only consists of labels A and B as the goal is to find the minimum number of splits between archaeal (A) and bacterial (B) genes. Thus, if the modified Fitch algorithm counts only one union operation, archaeal and bacterial proteins can be split by cutting a single branch in the tree. Additionally, only the number is recorded such that only the bottom-up phase is used, as depicted in Figure 53.

7.2.3 Comparison of Phylogenetic Trees

The reconstruction of phylogenetic trees based on the multiple sequence alignments obtained from Weiss et al. (2016) (SSC) resulted in 318 trees with a single split, 32 with two splits and 5 trees where three splits are needed to separate archaea and bacteria. The underlying clusters of proteins consists of on average 230 genes (minimum 9 genes, median 80 genes).

332 SSC were labeled with a corresponding COG, thus, we took the 332 COGs and reconstructed trees obtaining the set SSC^{COG}. However, SSC were significantly smaller as groups of genes in SSC^{COG} consisted on average out of 568 genes (minimum 8 genes, median 431 genes, also depicted in Table 5 and Figure 54 (right)). 35 SSC were assigned to more than one COG which results in a set of 293 distinct COGs. We reconstructed phylogenetic trees corresponding to these 293 COGs, as well as the groups of Harris et al. (2003) which consisted of 80 COGs, however, one of them consists of only bacterial sequences which was omitted in our analysis and we used 79 COGs when reconstructing

	total	single split	numbers of genes
SSC^{COG}	293	44	min 8, mean 568, median 431
Conserved COGs	50	47	min 238, mean 685, median 707
mixed COGs	2886	665	min 3, mean 348, median 241

Table 5: Sets of reconstructed trees based on different subsets of COGs. Values for the total number of trees, trees with a single split topology and numbers of genes are given.

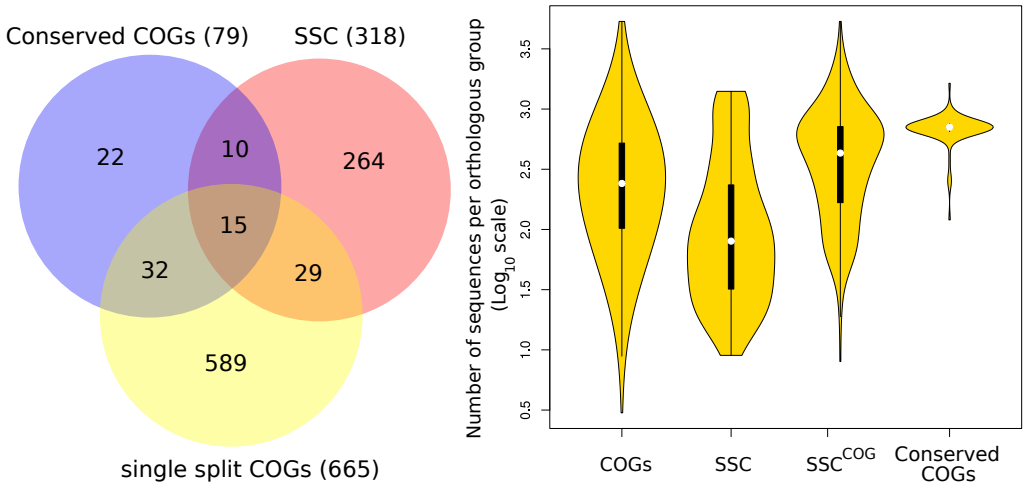


Figure 54: Venn diagram (left) showing overlapping COGs between the complete data sets of conserved COGs by Harris et al. (2003) (blue), the set of SSC by Weiss et al. (2016) (red) and the mixed COGs (yellow). For each group, there is a corresponding phylogenetic tree used to count the splits needed to separate archaeal and bacterial genes. Trees showing a single split are counted and overlaps are calculated based on the COG data set. Groups of Harris et al. (2003) and Weiss et al. (2016) claim to have a single split. The data set of mixed COGs is obtained by considering COGs which include genes from archaeal as well as bacterial genes and no paralogous sequences per group. Violin plot (right) depicting the number of sequences per group in $COGs$, SSC, SSC^{COG}, and the conserved COGs identified by Harris et al. (2003).

the tree for Harris et al. (2003). The single split COGs in SSC^{COG} are listed in Table 9 in Appendix B. A comparison of group sizes is also shown in Figure 55.

After analysis of the phylogenies for (SSC^{COG}), it was found that only 44 trees (or $\sim$ 15% of SSC) show a single split topology when more orthologs are included, (Figure 55, Table 9 in Appendix B). Here, around 41% of protein clusters belong to the functional category of information which is represented by only 15% regarding the set of COGs. Including COGs which exhibit up to 3 splits in their topology, 107 trees (or $\sim$ 28% of SSC) match with the reported tree topology of archaea bacteria separation reported previously. Only 25 protein families are common between the two previous studies, Harris et al. (2003) and Weiss et al. (2016), see Figure 54 (left). Phylogenetic trees drawn from the most recent version of the COGs showed overlaps with these previous study and also showed many more trees which split the bacterial and archaeal domains (in total 665 single split trees), as depicted in Figure 54 (right).

7.3 Interdomain vs Intradomain Distances

Phylogenetic analysis of the 50 single split trees obtained in Harris et al. (2003) with the most recent protein families from the COG database reveals that 47 of them show a single split given the current set of COGs (see Figure 57 and Table 5). This is remarkable, as

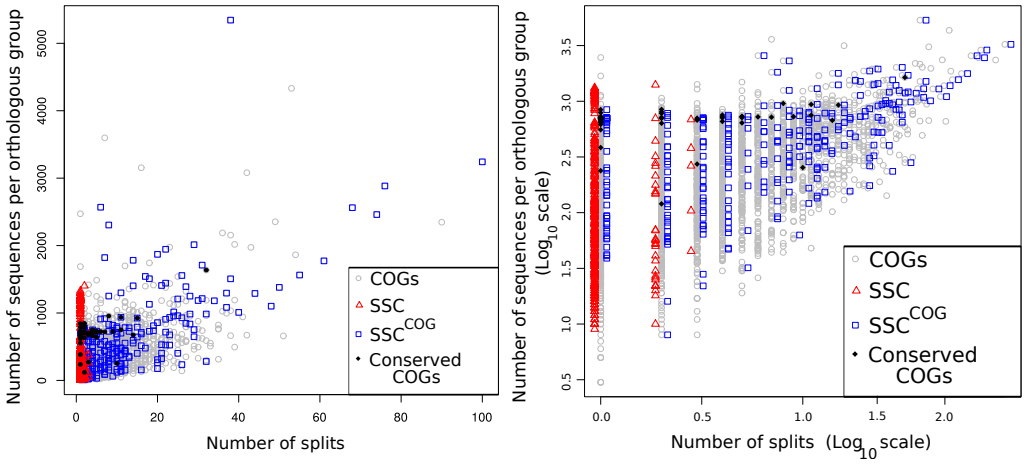


Figure 55: The number of sequences per group plotted against the number of interdomain splits found when the sequences are subjected to phylogenetic analysis in normal (left) and log_{10} scale (right). Single split clusters (SSC) from the previous study are in red, and the corresponding COG sequence populated families are in blue (SSC^{COG}), the complete set of COGs is shown in gray and the set of conserved COGs by Harris et al. (2003) is shown in black.

the study was conducted over 15 years ago, and made use of only 34 genomes. They used in total 80 COGs and found 50 to be well conserved. We reconstructed 79 of the 80 trees based on the current set of COGs and noticed that 15 of the 79 conserved COGs overlap with single split clusters of Weiss et al. (2016).

The number of interdomain splits observed in phylogenies is related to the number of sequences within a protein family (Figure 55), and we sought to develop a metric which would be less sensitive to incomplete protein family identification and which would also capture evolutionary qualities thought to be associated with LUCA derived protein families. Long interdomain branches may be indicative of a protein family having been in the LUCA, when the tempo of evolution was rapid, whereas newer families may have shorter branches separating the domains (Forterre, 2006; C. Woese, 1998). In line with this, we developed a metric $\overline{D}$ which describes the ratio of intra-domain to inter-domain phylogenetic distances found in a tree and applied it to phylogenetic trees derived from the COGs.

7.3.1 Calculating Tree Distances

In our own analyses we calculated the following values for each tree: the number of splits (s) needed to separate archaeal (A) and bacterial (B) genes and values for $\overline{D}$ based on pairwise distances between leaves in the tree. Distances between sequences can be calculated by summing up branch lengths on the path between pairs of leaves of the tree. We assume a tree to show a single split when genes of at least one of the domains are closely connected, thus average pairwise distances are relatively small inside this domain. Therefore, we calculated mean phylogenetic pairwise distances between leaves for

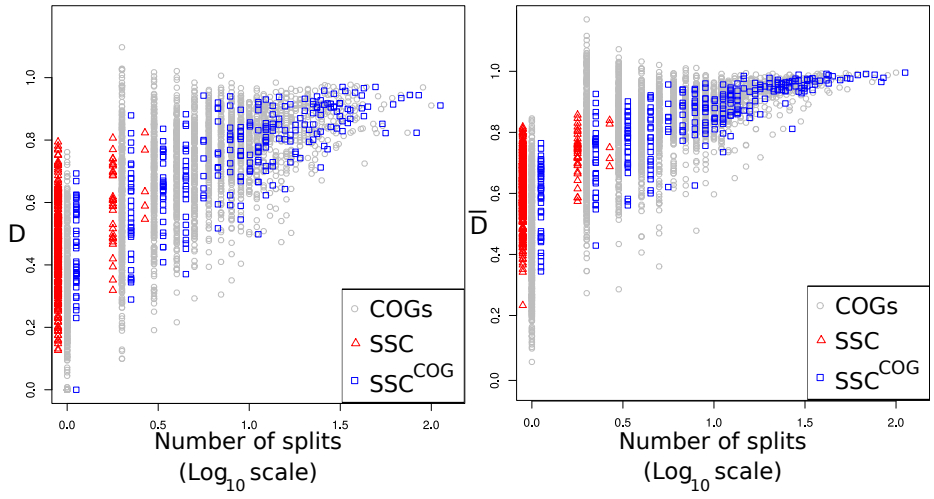


Figure 56: Left: D plotted against number of splits in log10 scale, for the data sets of COG (gray), SSC (red) and SSC^{COG} (blue). Right: $\overline{D}$ plotted against number of splits in log10 scale, for the data sets of COG (gray), SSC (red) and SSC^{COG} (blue).

intra-domain genes (between only archaeal or only bacterial sequences) and inter-domain distances, that is, the distances between archaeal and bacterial species. The following formulas show how calculations were conducted. Here, A is the set of archaeal and B the set of bacterial genes in a tree, with sizes n and m, respectively. The function $d_t(a_i, a_j)$ calculates the distance in the tree t between archaeal genes a_i and a_j and analogously for $d_t(b_i, b_j)$, for all $a_i, a_j \in A$ and $b_i, b_j \in B$. Then, $\overline{d}_{AA}(t)$ ($\overline{d}_{BB}(t)$) is the mean pairwise distance between archaeal (bacterial) species in tree t.

$$\overline{d}_{AA}(t) = \frac{\sum_{i,j=1}^{n} d_t(a_i, a_j)}{n \cdot (n-1)}, \quad \overline{d}_{BB}(t) = \frac{\sum_{i,j=1}^{m} d_t(b_i, b_j)}{m \cdot (m-1)}$$

The same can be done in order to calculate distances between genes from different groups, thus $d_{AB}(t)$ gives the mean pairwise distance between inter-group genes for tree t.

$$\overline{d}_{AB}(t) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} d_t(a_i, b_j)}{n \cdot m}$$

For each tree t, there is a set of genes for archaea and a set of genes for bacteria. We calculate the distance between each archaeal gene a and each bacterial gene b by summing up over all archaeal and bacterial genes in order to get every possible pair. Thus, the first sum takes all the archaeal genes in total n genes, and the second sum takes all bacterial genes of size m. As the value is dependent on the tree t, we indicate this by writing d_t. These distances can now be used to calculate the ratio of how closely related genes in one group (intra-group) are in comparison to inter-group distances, expressed by the value of $\overline{D}$.

$$\overline{D} = \frac{1/2 \cdot (\overline{d}_{AA} + \overline{d}_{BB})}{\overline{d}_{AB}}$$

A further possibility is to only consider the group of genes that has closer mutual relationships replacing the mean value by the minimum:

$$D = \frac{\min(\overline{d}_{AA}, \overline{d}_{BB})}{\overline{d}_{AB}}$$

Values for $\overline{D}$ are always larger than the corresponding D value.

Values for $\overline{D}$ are plotted in Figure 57, and comparisons for D and $\overline{D}$ values are shown in Figure 56 and Figure 58. In Figure 57, we highlighted COGs that were used in Harris et al. (2003) and Catchpole and Forterre (2019). We additionally calculated split and $\overline{D}$ values for trees obtained from MSAs of Weiss et al. (2016), depicted in Figure 56. Trees as depicted in Figure 52 were visualized using `iTOL` (Letunic and Bork, 2016). Values for D and $\overline{D}$ are also denoted as `Dmin` and `Dav` in the supplemental tables, respectively.

7.3.2 Classification of Phylogenetic Trees

The 3 groups of genes analyzed by Catchpole and Forterre (2019) illustrate the utility of this metric (depicted in Figure 57 by labeled dark green symbols). They analyzed the RNA polymerase beta subunit (RpoB) (COG0085, $\overline{D} = 0.27$), elongation factor G (COG0480, $\overline{D} = 0.44$) and reverse gyrase (COG1110, $\overline{D} = 0.85$) families (listed in Table 10 in Appendix B). COG1110 depicts only a portion of the tree reconstructed in Catchpole and Forterre (2019) and shows only two branches separating the archaea and bacteria domains (Figure 52). However, the calculated $\overline{D}$ value is relatively high, inconsistent with the family having been in the LUCA and suggesting a more modern protein family. Thus, $\overline{D}$ values appear to supplement phylogenetic inferences based on the analysis of domain separation, even in the case of incomplete phylogenetic sampling.

Applied to phylogenetic trees drawn from all the COGs, the most domain separated trees (low $\overline{D}$ values) contain a single split between archaea and bacteria groups (Figure 57). Protein families which display one split and low $\overline{D}$ values include some familiar proteins, for example: ribosomal protein S12 (COG0048, $\overline{D} = 0.15$, Figure 52), translation elongation factor Efp (COG0231, $\overline{D} = 0.24$), DNA-RNA polymerase RpoB and C (COG0086, $\overline{D} = 0.25$ and COG0085, $\overline{D} = 0.27$). Consistent with the finding of variable ages of ribosomal protein components (Kovacs et al., 2017), the ribosomal proteins do not have a coherent $\overline{D}$ value associated between them. However the small ribosomal subunits 12 (COG0048), 15 (COG0184), 2 (COG0052), 4 (COG0522), and 11 (COG0100) all have lower $\overline{D}$ values than any large subunit protein, listed in Table 8 in Appendix B. The protein families with low $\overline{D}$ values also overlap significantly with the nearly universal trees (NUTs) (Puigbò et al., 2009), indicating that conservation, domain separation, and long interdomain branches all coincide (Figure 58). In Puigbò et al. (2009) a subset of COGs was analyzed after reconstruction of corresponding phylogenetic trees and comparison of tree topologies. Here, Puigbò et al. (2009) develop a separation score indicating how well archaeal and bacterial genes are separated in the tree. A score of one shows perfect separation, thus, only one split would be needed to separate domains. Scores lower than one indicate horizontal gene transfer (HGT) events in the tree. The plot in Figure 58 shows data for 57 NUTs with a separation score of one (darkgreen) and 45 NUTs with a separation score between 0.62 and 0.92 (purple).

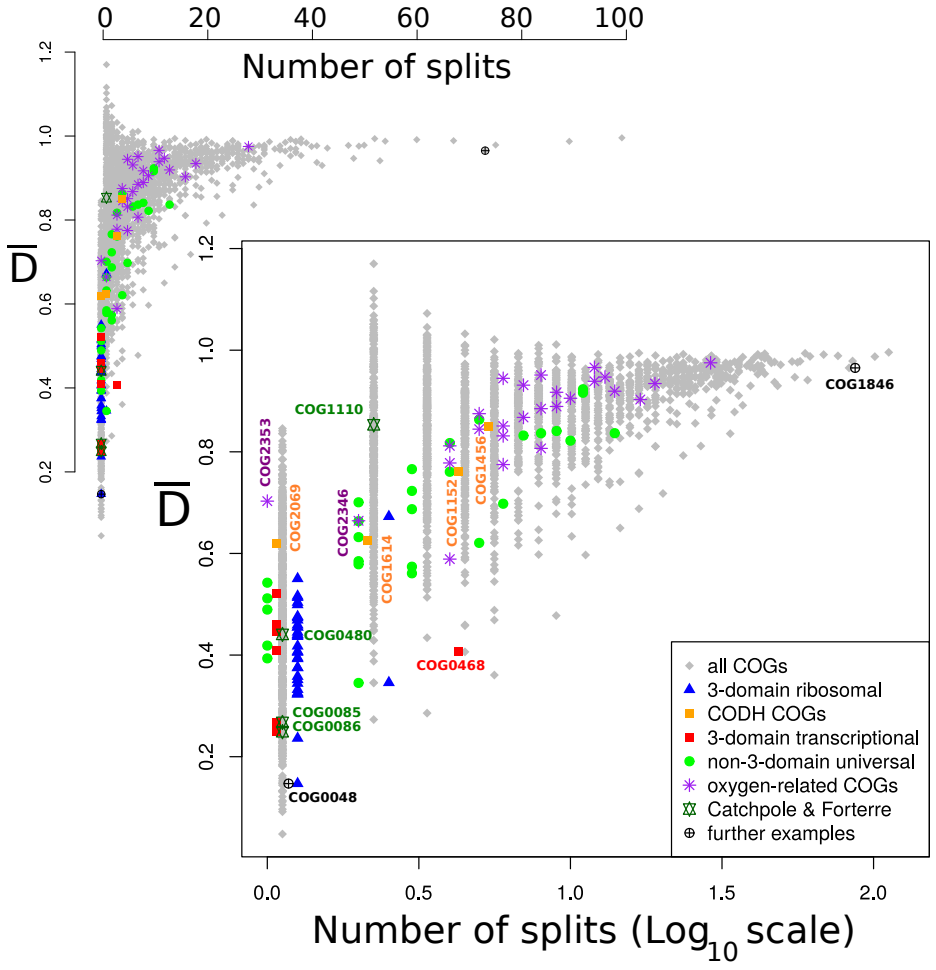


Figure 57: Interdomain split values for each COG plotted against $\overline{D}$. For visualization, the logarithm was used to display split values where the inset shows distribution in normal scale. Symbols are slightly shifted to avoid overlays, and the colored symbols indicate subgroups as defined by Harris et al. (2003), Catchpole and Forterre (2019), oxygen related COGs, CODH/ACS COGs and other examples discussed in the text.

COGs associated with oxygen metabolism (S. Liu et al., 2018) all appear to have similar inter:intra-domain phylogenetic distance ratios ($\overline{D} \sim 0.5$), and similar results were obtained for COGs comprising the four subunits of the CODH/ACS enzyme complex, which are thought to be associated with the LUCA or ancient interdomain LGT events (Adam et al., 2018; Inoue et al., 2019), as depicted in Figure 57. Indeed, protein families involved in metabolic processes seem to not only be susceptible to lateral gene transfers, but they also do not display long domain separating branches, e.g. COG0636 (the Na+ binding c subunits of the ATP synthases ($s = 10$, $\overline{D} = 0.82$), COG1740 (the [Ni-Fe]

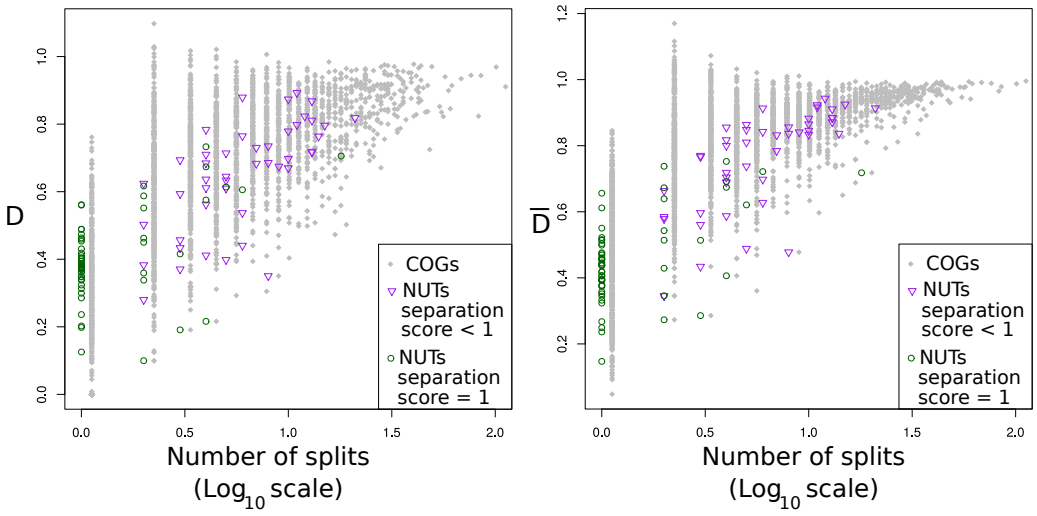


Figure 58: Plots comparing number of splits in log10 scale to D (left) and $\overline{D}$ (right) for the set of COGs (gray), the set of NUTs with a separation score equals 1 (darkgreen) and a separation score below 1 (purple) indicating HGT (Puigbò et al., 2009).

hydrogenase small subunit $s = 3$, $\overline{D} = 0.78$) and COG1229 (the Formylmethanofuran dehydrogenase subunit $s = 2$, $\overline{D} = 0.63$) (see Table 8 in Appendix B). It appears that the amount of within domain evolution is about the same as between domain evolution in these cases, which is clearly different than in the case of the single split, low $\overline{D}$ value proteins.

7.4 Permutation Analysis

During reconstruction of phylogenetic trees, leaves were labeled by taxon identifiers and marked A or B to indicate the species belonging to the domain of archaea or bacteria. In order to create a randomized reference set of trees, domain identifiers were shuffled such that tree topology and size were kept and for each leaf, we randomly chose to set the label to A or B. This resulted in three data sets with the following percentages of A and B labels: (i) 30% A and 70% B, (ii) 50% A and 50% B, (iii) 90% A and 10% B. For each of the trees in the randomized data sets, the number of splits and values for D and $\overline{D}$ were calculated. Figure 59 shows distribution of the number of splits, values for $\overline{D}$ and size of groups for the three randomized data sets in comparison to the set of COGs.

Permutation analysis reveals that trees drawn from biological datasets are significantly different from random sampling iterations: for trees made from multiple proportions of simulated archaea and bacteria, only small (less than 10 genes per group) trees showed a single split, and $\overline{D}$ values do not decrease below 0.51 for these single split trees (see also Figure 59). The distribution of $\overline{D}$ values for the COGs shows that intra-group distances are mostly smaller than inter-group distances which is not the case for the randomized

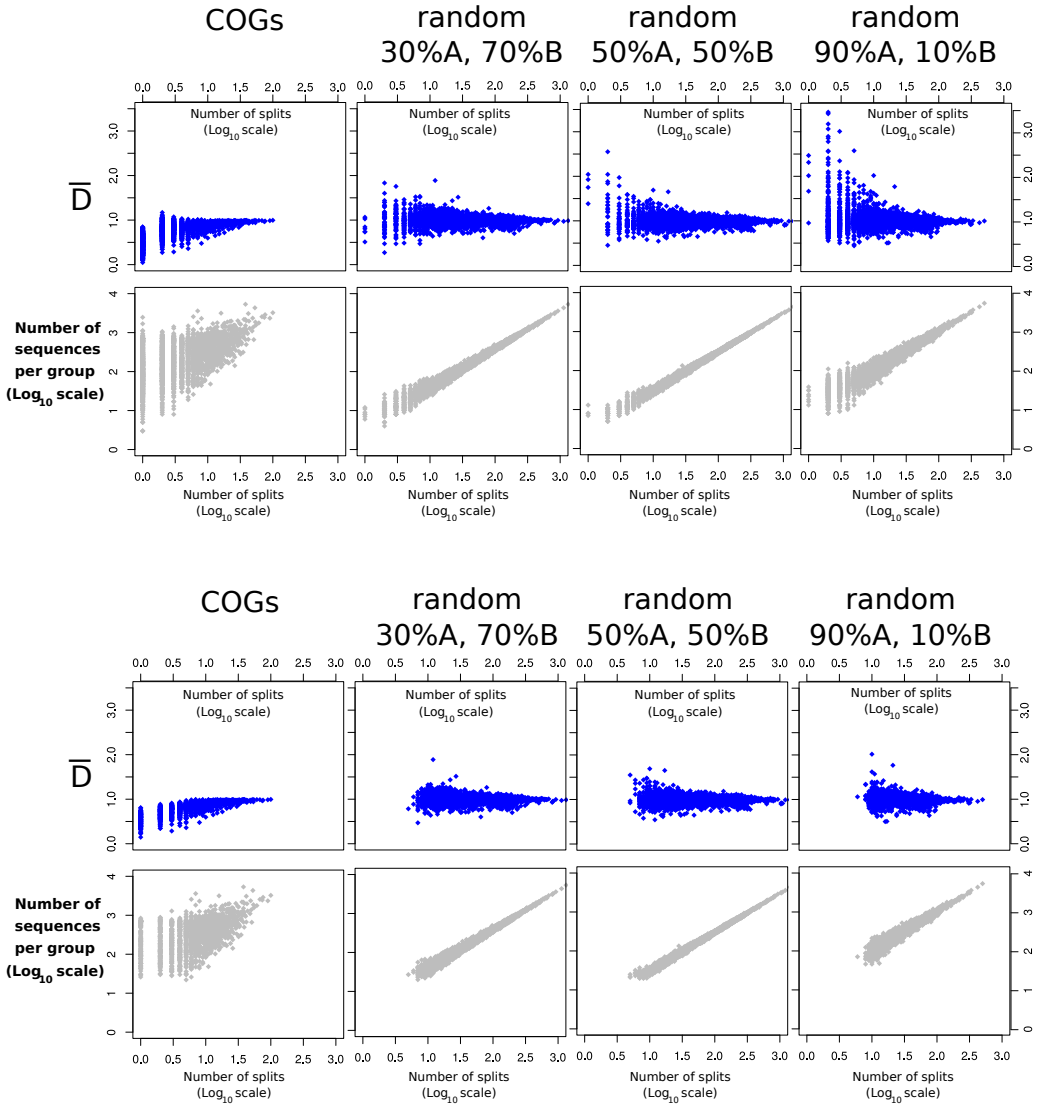


Figure 59: Plots comparing number of splits in log10 scale to $\overline{D}$ (blue) and the number of sequences per group in log10 scale (gray) for the set of COGs and randomized trees including all trees (top) and trees with at least 10 leaves of each label, a and b (bottom).

trees. It can be seen that $\overline{D}$ values are mainly distributed between 0 and 1 for COGs whereas the distribution is significantly shifted upwards for the randomized trees.

The plots also underline the importance of the correlation between the number of genes per domain and the number of splits. The shuffled trees suggest that it is more probable to get randomized tree topologies with low number of splits if the trees are small

or the proportion of the amount of genes within each domain is unbalanced. This leads to the assumption that COGs with a higher number of genes (in both domains) provide more reliable data than smaller gene clusters. Of course, a well-distributed set of genes over the tree of life is fundamental.

As depicted in Figure 59 (bottom), there are no trees with less than 5 splits for the randomized trees. For trees with at least 10 genes per group, the minimal number of splits is 5 which stands in contrast to the set of COGs where 128 single split trees can be found. This clearly shows that a low number of splits highly correlates with a low number of sequences per group. For the COGs with at least 10 genes in each domain, we get a median of 7 splits. For the randomized trees, median values of splits are 29 (90% A, 10% B), 55 (50% A, 50% B) and 47 (30% A, 70% B).

7.5 Concluding Remarks

We have presented several ways of reconstruction evolutionary history based on sets of orthologous genes for archaeal and bacterial species. Despite an agreed upon definition of orthology, methods of creating both sets of orthologous genes and for the reconstruction of phylogenetic trees differ significantly. We discuss and compare results of different studies with our own adapted measure in order to gain insights into advantages and disadvantages of different methods. While Harris et al. (2003) applied strict parameters within their study (those COGs conserved in the 34 genomes analyzed at the time), their results include a relatively small set of only 80 sequences assumed to be conserved since lowest universal common ancestor (LUCA). In contrast, Weiss et al. (2016) did not include conservation selection criterion and as such that their set of LUCA genes has a larger size (355).

As a criterion of gauging whether or not a protein family was in the LCA, conserved presence in a genomic "core" present in all taxa provides limited insight into the traits of the LUCA, because so few protein families are conserved (Koonin, 2003; Charlebois and W. F. Doolittle, 2004; Puigbò et al., 2009). Relaxing the requirement of conservation allows collection of more LUCA candidates which can then be assessed by their phylogenetic character (Harris et al., 2003; Puigbò et al., 2009). Further work is needed to identify the functional attributes of proteins found within diverse families. For example, the phosphate acetyltransferase (Pta) found by Weiss et al. (2016) corresponds to a non-catalytic (C-terminal domain lacking) paralog of the full protein, meaning that the protein cannot function as imagined in that report. It will be beneficial to include structural information to better estimate phylogenetic distances and what constrains these distances. Geochemical data can give further clues about the environmental conditions on early Earth, allowing for phylogenetic-geochemical calibrations to be made (e.g. Wolfe and Fournier (2018) and Shih et al. (2017)).

CHAPTER 8

Unbiased Map of Transcription Termination Sites in *Haloferax volcanii*

Contents

Detailed analysis of cellular processes and corresponding genes will give new insights into early evolution, development of new species and environmental conditions on early earth. This chapter describes the transcription termination sites of the halophilic model archaeon *Haloferax volcanii*. *H. volcanii* has been used for a plethora of biological studies (Leigh et al., 2011; Pohlschröder and Schulze, 2019), including the determination of nucleosome coverage (Ammar et al., 2012) and a genome-wide identification of transcription start sites (TSS) (Babski et al., 2016). *H. volcanii* requires high salt concentrations for optimal growth, and due to the high intracellular salt concentrations, RNA-protein interactions -including modes of transcription termination- may differ from those in mesophilic archaea.

This chapter is based on Berkemer et al. (2020), titled *Identification of RNA 3' ends and termination sites in* Haloferax volcanii. For further details, see the full publication, the corresponding supplemental files and Appendix C. The code of the pipeline together with explanations and examples is available at Bioinformatics Leipzig (`http://www.bioinf.uni-leipzig.de/publications/supplements/18-059`).

In contrast to earlier chapters, the project described here is based on pure application of dynamic programming (DP) algorithms combined with self-implemented scripts to prepare and analyze the data. The analysis for transcription termination sites is based on transcriptome sequencing data that was mapped to the reference genome (see Subsection 5.4.2 for background on sequencing and mapping and Appendix C for details of mapping the data set for TTS). Resulting genomic positions were checked for flanking sequence or secondary structure motifs (see also Chapter 5). This was done by sequential application of various algorithms and pipelines.

An example for a transcription termination site is shown in Figure 60. This transcription termination site (TTS) is based on the DSM analysis (Dar et al., 2016a) and shortly described in Section 8.2. The TTS has also been detected with the IE-PC approach which is described in detail in this chapter and in Berkemer et al. (2020). A comparison of both approaches will be given in Subsection 8.4.1. Figure 60 nicely shows the peak in read end coverage just downstream of the annotated gene pilA2 indicating the transcription termination site.

8.1 Transcription Termination in Archaea

Whereas some data have been reported on transcription initiation and elongation in archaea, very little is known about transcription termination (Maier and Marchfelder, 2019). Controlled transcription termination is important to avoid aberrant RNA molecules and to help with RNA polymerase recycling. Generally, the genes in archaeal chromosomes are densely packed, so that proper termination is also important to prevent transcription from continuing into downstream genes. The process of transcription termination is not trivial because the very stable transcription elongation complex must be destabilized and dissociated during termination. In bacteria, two major classes of termination signals have been described: intrinsic termination and factor-dependent termination (Ray-Soni et al., 2016; Porrua et al., 2016). Intrinsic termination occurs either at a stretch of Ts or at hairpin structures that fold in the newly synthesized RNA; both trigger dissociation of the elongation complex. Factor-dependent termination occurs upon interaction with a specific protein such as the bacterial termination factor Rho (Peters et al., 2011). Protein factor-

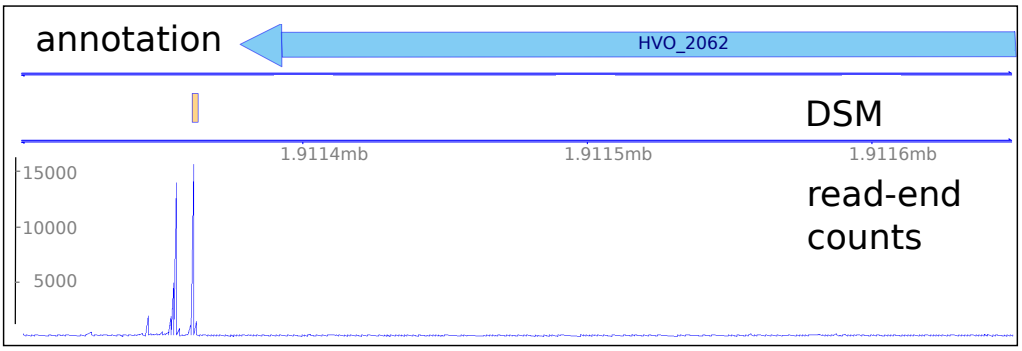


Figure 60: Termination site downstream of the pilA2 gene (HVO_2062). The termination site is indicated by an orange rectangle in the middle panel. The annotated gene is shown in the upper panel, and the genomic location is indicated below the annotation and above the coverage as a blue line with coordinates given in Mb. The read end counts are shown in the lower panel (read counts per position), and DSM data are reported as binary signals; thus, either a signal is present or not. The Figure was created using the `R package Gviz` (Hahne and Ivanek, 2016).

assisted termination is especially important in regions where strong selective pressure on the DNA sequence does not allow encoding of intrinsic termination signals. This may be the case when termination must occur in the coding region of the downstream gene. In eukaryotes, several RNA polymerases synthesize the different RNA classes, and each polymerase has different modes of termination, including protein factors (Németh et al., 2013), poly-T stretches (Arimbasseri et al., 2013) or further additional factors (Kuehner et al., 2011). Whereas archaeal transcription initiation resembles eukaryotic RNA polymerase II initiation (Fouqueau et al., 2017), transcription elongation and termination seem to be more similar to the eukaryotic RNA polymerase III pathway, which is independent of RNA secondary structures and protein co-factors (Santangelo and Reeve, 2006). Compared to the determination of transcription start sites, the identification of termination sites is more complex. Termination is often leaky and encompasses several consecutive sites, and degradation by exonucleases renders the 3' ends heterogeneous and less clear.

Archaeal genomes are densely packed, thus, correct transcription termination is an important factor for orchestrated gene expression, since faulty termination, for instance, can result in aberrant transcription of downstream genes. Data reported on archaeal termination so far show that intrinsic termination occurs with a run of Ts (Santangelo and Reeve, 2006; Hirtreiter et al., 2010; Santangelo et al., 2009; Spitalny and Thomm, 2008; Thomm et al., 1993) and is potentially also influenced by secondary structure elements (Santangelo and Reeve, 2006; Thomm et al., 1993). Factor-dependent termination was predicted based on the results of an in vivo reporter assay in the archaeon *Thermococcus kodakarensis* (Santangelo et al., 2008). A recent study confirmed this hypothesis, reporting the discovery of the first archaeal termination factor (Walker et al., 2017). Recently, the termination sites for two archaeal organisms (*Sulfolobus solfataricus* and *Methanosarcina mazei*) were investigated systematically using RNAseq data (Dar et al., 2016a) .

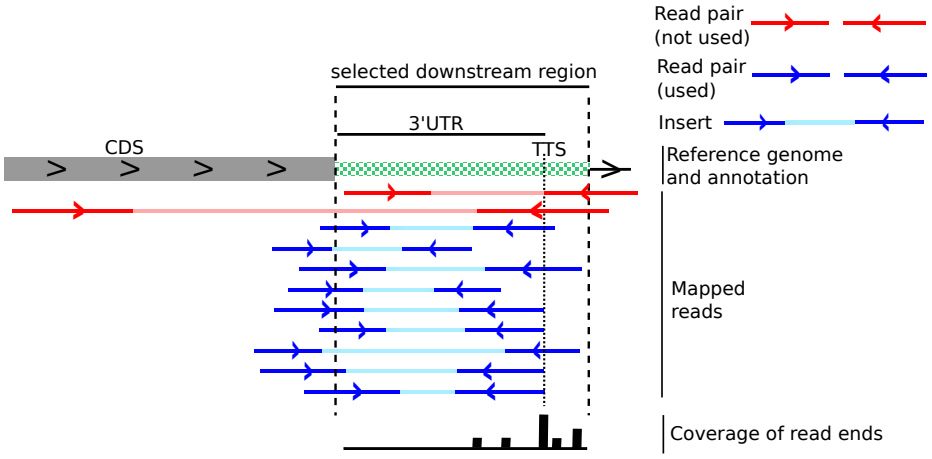


Figure 61: Principle of the Dar-Sorek-Method. As a first step in DSM, read pairs with an insert overlapping an annotated region are selected (red and blue lines)(Dar et al., 2016a). Inserts that do not overlap or have a length of more than 500 nucleotides are discarded (red read pairs in the Figure). From the selected read pairs (blue), the average length over all inserts is calculated as described in Dar et al. (2016a). This value is used to determine the length of the selected downstream region (green white region in the Figure). The coverage of all read ends in this region is retrieved (bottom line), and the position with the highest coverage is identified as the TTS.

8.2 Dar-Sorek Method

Mapped reads were initially analyzed using a self-implemented version of the method described by Dar et al. (2016a), which will be referred to as the Dar-Sorek-Method (DSM). This method identifies TTS in a defined region downstream of annotated genes at the position with the highest coverage of mapped read ends. The length of the downstream region is determined by the average insert lengths of corresponding paired-end reads (Figure 61). Numbers of detected TTS are shown in Table 12 in Appendix C. In our dataset, the median length of the analyzed region was 126 bp long. The resulting 3' UTRs were mostly shorter than 100 nucleotides, with a median length of 58 nucleotides. The length restriction given in the DSM approach might be too strict for genes with long 3' UTRs. In addition, the method cannot determine TTS independent of an annotation. Using DSM, we identified 3,155 termination sites for the complete Haloferax genome of which 85% were in intergenic and the remainder in coding regions (see Table 12). A typical termination site is shown in Figure 60 for the pilA2 gene (HVO_2062).

8.3 Internal Enrichment-Peak Calling

The length restriction given in the DSM approach might be too strict for genes with long 3' UTRs. In addition, DSM analysis only includes sequences downstream of annotated genes and, thus, only a fraction of the genome (Dar et al., 2016a). To overcome this

restriction, we developed a novel approach to interpret the RNAseq data obtained that we termed *Internal Enrichment-Peak Calling* (IE-PC).

To determine TTS, the DSM approach used reads from a total cellular RNA fraction that contains RNA 3' ends derived from transcription termination as well as 3' ends derived from processing. Thus the 3' ends identified by DSM are not all TTS but also processing sites (PS). A similar problem exists for the determination of original transcription 5' ends, where a well established method for the reliable identification of transcription start sites (TSS) has been developed, termed differential RNAseq (dRNAseq). Here, data derived from an RNA sample treated with terminator exonuclease (+TEX) are compared with data obtained from an untreated sample (-TEX). TEX treatment of an RNA sample enriches primary transcripts containing the original 5' triphosphate end. Comparison of data from the +TEX sample with the -TEX sample helps to identify TSS (Babski et al., 2016). We used the dRNAseq approach to determine original termination ends. RNA still containing their original 5' end have a higher probability to also still contain their original 3' end. Therefore, we isolated -in addition to the total RNA fraction already obtained- a second fraction that was treated with 5' terminator exonuclease (TEX) to enrich primary transcripts and thereby original termination ends. After cDNA library generation from the TEX treated RNA, NGS was performed, resulting in an average of 40 million reads for each of the three libraries. Details on the numbers of mapped reads can be found in Table 11 in Appendix C.

Internal Enrichment requires mate pair sequencing data, from which a set of read pairs is selected that contains the same original first-strand cDNA 5' ends (Figure 62). The corresponding 3' ends of the mate pair reads are then evaluated. Sequencing the cDNAs generated from these first-strand cDNA fragments in paired-end mode preserved information about fragment ends. After mapping the read pairs to the reference genome, the hallmark of an original RNA 3' end was its high coverage of read ends, with its associated mate ends originating from a multitude of genomic sites. It is highly unlikely that independent clones end at an identical position (Figure 62). Multiple fragments with a heterogeneous 5' end but a common 3' end thus are indicative of likely TTS candidates. However, this method cannot distinguish between primary termination sites and RNA processing sites. Therefore, a peak calling (PC) step was added. The two methods (IE and PC) were sequentially run on the data, and only sites that were found by both approaches, IE as well as PC, were considered to be bona-fide TTS. Both methods will be explained in the subsequent subsections. We allowed a maximal distance of 10 nucleotides when computing overlapping sites for IE and PC. The advantage of this algorithm is that it is independent of genome annotation and thus analyses the complete genome sequence rather than a restricted region allowing the identification of all TTS of a genome. This resulted in a set of putative primary TTS detected based on +TEX data and a second and larger set resulting from IE-PC applied on the -TEX data.

Internal Enrichment

Due to the library preparation, native 3' ends of transcripts should be enriched in the sequenced +TEX RNAseq library similar to the enrichment of primary transcript 5' ends (Figure 62). To detect sites with a significant enrichment of sequenced and mapped fragment ends, a sound background without enrichment is desired. In the current setting,

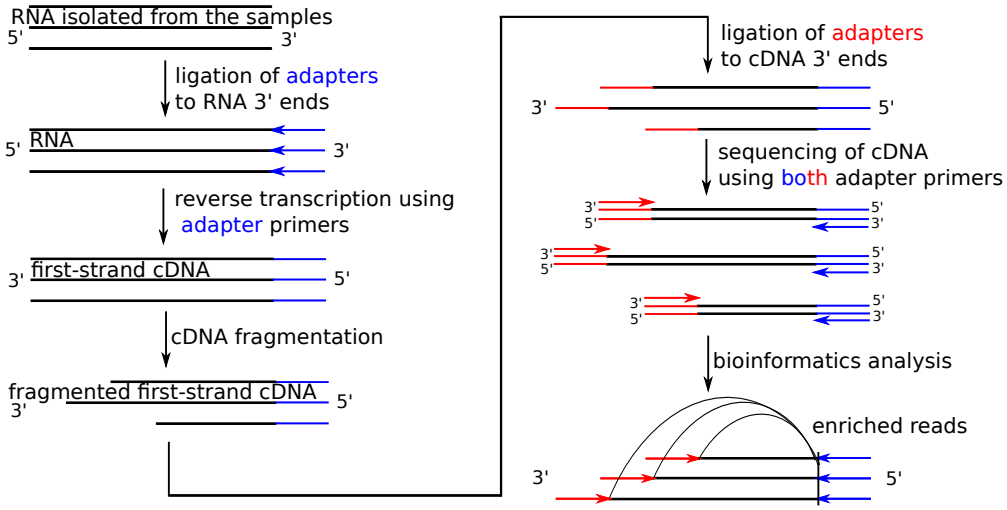


Figure 62: Principle of internal enrichment. After RNA isolation, adapter primers were ligated to the RNA 3' end and the RNA was reverse transcribed. First-strand single-stranded cDNA was fragmented prior to the addition of the adapter primer at the cDNA 3' end. The break points were considered to be random, leading to an enrichment of original RNA 3' ends over 5' fragmentation ends. Sequencing the cDNAs generated from these first-strand cDNA fragments in paired-end mode preserved information about fragment ends, even if they were longer than the read length. After mapping the read pairs to the reference genome, the hallmark of an original 3' end was its high coverage of read ends, with its associated mate ends originating from a multitude of genomic sites.

we used the intrinsic properties of a paired end sequencing run to directly deduce the following information. Since each fragment which results from an individual fragmentation event (in contrast to PCR duplicated fragments) is very unlikely to have the exact same length, truly enriched sites can be expected to be associated with sequenced fragments all ending at the respective site but starting at different positions. Therefore, the more different mates (mapping to different positions) are associated with the different reads ending at a particular site the higher the enrichment of read end signal at that particular position can be considered. To capture this, we calculate for each position i a score S as

$$S_i = \frac{C_i}{(\prod_{j=1}^{n} C_j)^{1/n}} \; \forall j \; \exists R(i \circ j)$$

Thereby, C_i denotes the number of fragment ends at position i, C_j the number of fragment starts at position j, and $R(i \circ j)$ all positions i, j which are associated via at least one read-mate pair R. To get an expected background distribution of these scores, we again use the nature of paired-end reads. Since we expect only fragment ends, in contrast to fragment starts, to be enriched, we can use the distribution of the reciprocally defined scores for the fragment start S_j as a background distribution.

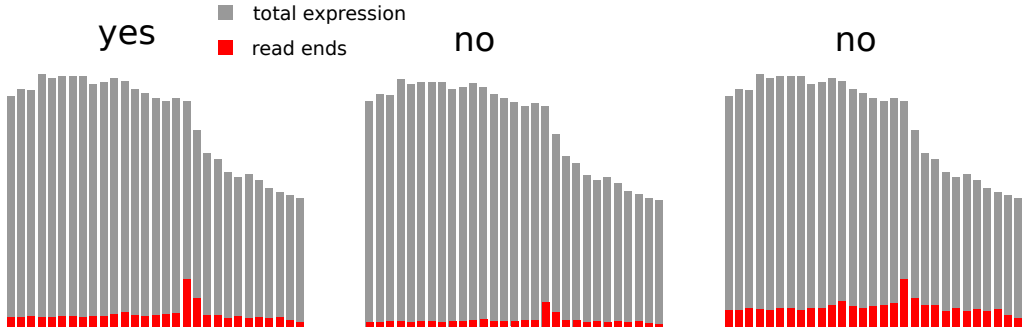


Figure 63: Principle of the applied peak calling procedure. In a sliding window approach, positions where the number of read stops (red) exceeded the mean number of read stops over the whole window by a z-score above the threshold 2 were treated as potential endings (left), while potential peaks of the same height but below a z-score of 2 were discarded (right). In addition, if the number of read stops was below a 10% threshold relative to the total coverage (grey), the peak was discarded (center).

Peak Calling

A complementary approach to find transcription termination sites is identifying peaks in read stops (Figure 63). In order to find these peaks, we first computed the strand specific read coverage at every position in the genome, which we used as a background. We then used a sliding window approach with a window size of 150 nt and an overlap of 50 nt to find positions in the respective windows with a significantly higher number of read stops than the rest of the window. This was done by computing the mean number of stops as well as the standard deviation by window and then calculating the z-score for the number of read ends at every position of the window. Subsequently, we required a minimum number of 5 reads stopping, at least 10% of all reads covering position $i-1$ must end at position i as well as a minimum z-score of +2 at a site to report it as a putative TTS. As a consequence of the overlapping of the windows, a site is evaluated multiple times and must fulfil the criteria in at least one contexts evaluated.

8.4 IE-PC results

Application of the IE-PC algorithm to the data from the TEX treated sample consisting of enriched original transcription termination sites identifies 1,543 TTS (Table 6), whereas 6,284 RNA 3' ends are identified when using the -TEX data set. 1,220 TTS are found in both data sets. The following results are based on the small, conservative analysis from the +TEX data set, that contains enriched TTS.

Inspection of the TTS obtained revealed very closely spaced TTS, that were less than 10 nucleotides apart. In addition, we found a relatively high fraction of closely spaced TTS, that were 11 to 150 nucleotides apart. The closely spaced TTS were subclassified into primary TTS and non-primary TTS, a primary TTS is located directly downstream of a 3' gene end on the same strand (Figure 64 A & B). A non-primary TTS is located

Data sets	TTS (+TEX)	TTS & PS (-TEX)	TTS (both)
intergenic	807	2,856	670
coding	736	3,428	550
total	1,543	6,284	1,220

Table 6: TTS and PS identified with IE-PC. The IE-PC algorithm was run on two data sets: data from a TEX treated sample (column +TEX) as well as on data from an untreated RNA fraction (column -TEX). Most of the original TTS identified in the +TEX data are also present in the data set from the untreated RNA fraction (column TTS found in both fractions).

downstream of another TTS, with no other features (like TSS or 3' gene end) in between, as shown in Figure 64 C. Very closely spaced TTS in a region of up to 10 nucleotides were assumed to reflect stuttering of the RNA polymerase. Thus, this was reported as only a single TTS (the one with the highest coverage). It should be noted that stuttering may happen several times, so that the overall length covered by such a termination region may significantly exceed 10 nucleotides. The TTS in the "stuttering region" were grouped into multiplets and termed termination regions. For the +TEX sample we found in total 1,056 primary TTS and 487 non-primary TTS, for the -TEX sample, we detected 1,944 primary and 2,486 non-primary TTS.

As expected, the IE-PC algorithm employed on the +TEX data set identified less TTS than with the -TEX data set and less than DSM, that was also run with a -TEX data set. Altogether, 1,543 transcription termination sites were found (Table 1). Slightly more than half of the sites were found in intergenic regions (807 TTS, 52%) and the remainder in coding regions (736 TTS, 48%). Detailed information for each TTS can be found in the corresponding supplementary data of Berkemer et al. (2020). Analysis of the regions up- and downstream of the TTS were performed separately for TTS located in coding and intergenic regions (Figures 65a and 65b), and in both an increase in hybridization energy at the TTS similar to the increase identified in the TTS set obtained with DSM was found. The pattern of nucleotide enrichment for sites located in intergenic regions showed that Ts were prevalent at the TTS (Figure 65a). To calculate hybridization energies, `RNAplfold` (Lorenz et al., 2012) (`Vienna RNApackage 2.0`, Lorenz et al. (2011b)) was used.

We next analyzed sequences 15 nucleotides up- and five nucleotides downstream of

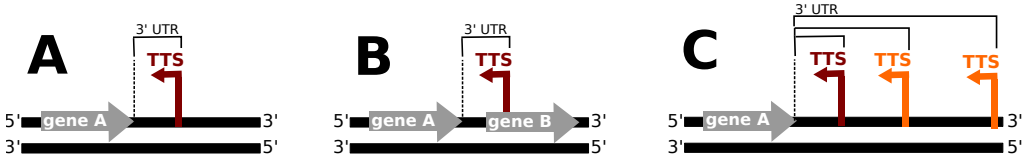
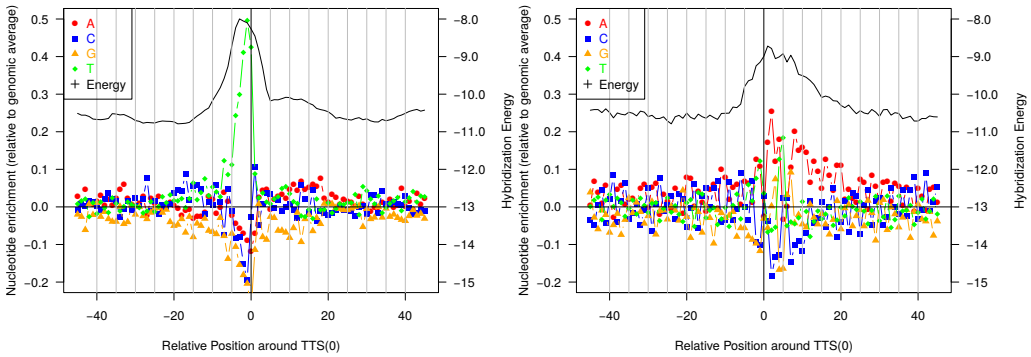


Figure 64: Location of TTS. A. TTS is located in an intergenic region. B. Location of the TTS in an annotated gene. C. A primary (red) and two non-primary TTS (orange) are shown. The length of the UTR was in all cases measured as the distance between the TTS and the 3' end of the upstream annotated gene on the same strand.



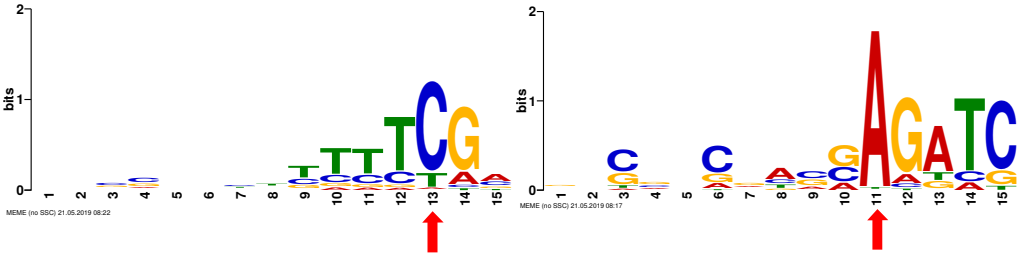
(a) Nucleotide enrichment in intergenic regions. **(b)** Nucleotide enrichment in coding regions.

Figure 65: Analysis of up- and downstream regions for TTS identified with IE-PC. Intergenic (Figure 65a) and coding (Figure 65b) regions were investigated. Forty-five nucleotides up- and downstream of the termination site were analyzed for (1) nucleotide enrichment at each position (left y-axis) and (2) the hybridization energy (right y-axis). x-axis: nucleotide position (upstream -, downstream +). The color scheme for the four nucleotides is shown at the upper left, and the energy data are shown with a black line. Hybridization energies were calculated based on the binding energies between the DNA template and RNA in the area behind the RNA polymerase.

the 807 intergenic termination sites for common sequence motifs. For 748 sites, we found similarities in the sequences, such as a conserved dinucleotide TC as part of the motif as well as a stretch of T's of variable length upstream of the termination site (Figure 66a). Sequences 15 nucleotides up- and five nucleotides downstream of the 736 TTS in coding regions were likewise investigated for common sequence motifs (Figure 66b). The prominent C residue at every third position is typical for coding regions in *Haloferax*. The third codon position combines an enrichment for GC (due to the GC-rich genome) and pyrimidines. For 456 TTS in coding regions, we found the downstream motif AGATC (Figure 6B). Taken together, we could identify distinct termination motifs that were specific for intergenic and for coding regions. Motifs detection was conducted using MEME (Bailey et al., 2009) (version 5.0.1). Taken together, we could identify distinct termination motifs that were specific for intergenic and for coding regions. The same motifs could be found for TTS in the -TEX data set. See supplemental data of Berkemer et al. (2020) for more details.

8.4.1 Comparison DSM and IE-PC

Our newly established tool IE-PC is able to determine TTS covering the complete genome and thus can identify all TTS of an organism. Consequently our tool identifies more TTS than the DSM tool. Comparison of signals obtained with both methods can only be done with the regions that are included in the DSM analysis. The original DSM analysis was based on regions with a median length of 126 nucleotides downstream of an annotated 3' end, with the TTS located at the position with the highest coverage. We further compared individual TTS found with DSM and/or IE-PC in downstream regions defined by the DSM analysis in more detail. Examples are shown in Figure 67, and Table 7 lists the



(a) Area around TTS in intergenic regions. **(b)** Area around TTS in intergenic regions.

Figure 66: Enriched sequence motifs around the TTS identified with IE-PC. (a) Sequence motif close to TTS located in intergenic regions. The TTS is located at position 13 (red arrow). (b) Sequence motif close to TTS located in coding regions. The TTS is located at position 11 (red arrow). Motifs were detected using MEME (Bailey et al., 2009).

number of TTS that were identified by both methods (allowing for discrepancy of up to 10 nucleotides) as well as TTS that appeared in only one of the data sets. Altogether, we found 1,664 TTS being present in both data sets (Table 7).

For the genes *trmY* and $tRNA^{Pro}$, both methods identified one identical TTS position and two different TTS positions (Figure 67). Here, the lower panel (DSM) shows the results obtained with DSM, coverage is shown as the 3' end coverage of corresponding reads, the TTS position for the transcript is also shown. The upper panel (IE-PC) shows IE-PC results, with TTS location and read coverage of the -TEX data set above. The drop in coverage values is clearly visible, matching the TTS location as identified by IE-PC. Corresponding read end coverage of the DSM data set was present at the same position. Figure 67 shows two genes on the forward strand, *trmY* (HVO_1989) and $tRNA^{Pro}$ (HVO_1990). The IE-PC analysis identified two TTS downstream of the tRNA-Pro gene (upper panel). The corresponding coverage values for DSM (lower panel) showed similar signals, but due to its specific algorithm, DSM assigned one TTS downstream of the *trmY*

chromosome	IE-PC only	overlapping	DSM only
CHR	3,184	1,296	1,128
pHV1	231	53	39
pHV3	394	97	125
pHV4	811	218	199
total	4,620	1,664	1,491

Table 7: Comparison of TTS found with DSM and IE-PC in downstream regions covered by DSM. The column 'IE-PC only' lists all TTS identified by IE-PC but not with those identified by DSM method. TTS identified with IE-PC that were also found with DSM are listed in the column 'overlapping'. Sites that were detected in the defined region only by DSM and that did not overlap with IE-PC sites are listed in the column 'DSM only'. The IE-PC data shown here are the ones calculated on the basis of the -TEX data set, since the DSM data are also based on the -TEX data.

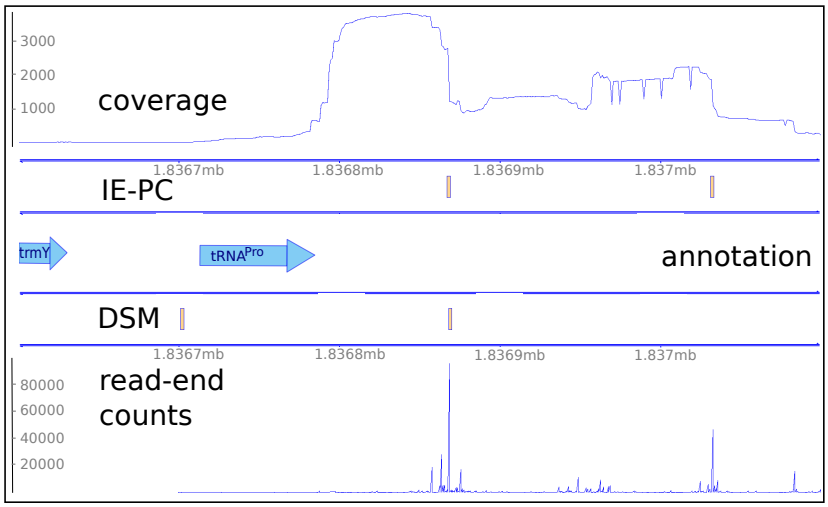


Figure 67: TTS comparison of DSM and IE-PC for the $tRNA^{Pro}$ gene. In the lower panel, termination sites and corresponding read end coverages from the DSM data set are shown, assigning a TTS downstream of the *trmY* gene and another one downstream of the $tRNA^{Pro}$ gene (shown as orange rectangles). The upper panel shows the TTS location determined by IE-PC and the total coverages, corresponding to the -TEX data set, identifying two TTS downstream of the $tRNA^{Pro}$ gene. Since sequencing starts at the 3' end, coverage starts at the 3' end and runs continuously for 75 bp due to the read length. The secondary TTS of the $tRNA^{Pro}$ gene was not reported by the DSM algorithm as this algorithm systematically reports only a single TTS for each annotated gene.

gene (HVO_1989) and one downstream of the tRNA-Pro gene (lower panel). Since in the DSM analysis, every gene had a downstream region with an individual length, by calculating for each gene the average insert length of the corresponding reads. Using DSM, the length for the downstream region of HVO_1989, was calculated with only 110 nucleotides, for HVO_1990 the result was 145 nucleotides. Therefore, the high read end coverage in the right part of the figure was beyond the region selected for analysis.

8.4.2 Secondary structures can act as termination signals

To identify potential secondary structure motifs, we conducted a search for accessible and inaccessible regions around the TTS using `RNAplfold` (Lorenz et al., 2012) (`Vienna RNApackage 2.0`, Lorenz et al. (2011b)). We plotted accessibilities against a background distribution of shuffled dinucleotides. However, no clear signals were found to indicate significantly increased or decreased accessibility.

In a second attempt to identify secondary structures, we applied `Graphclust 2.0` (Miladi et al., 2017; Miladi et al., 2019) within `Galaxy` (Afgan et al., 2018) to sequences 100 nucleotides upstream of all TTS. `Graphclust` is a tool that clusters input sequences based on their secondary structure(s). It will cut the sequences based on a window size

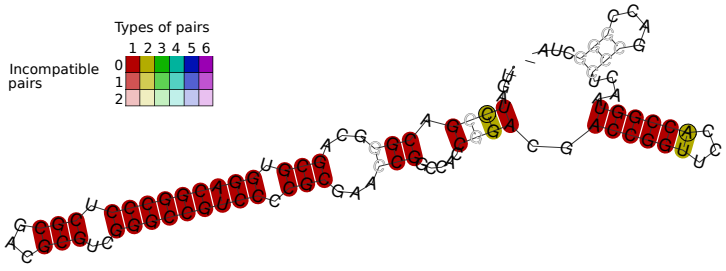


Figure 68: Hairpin structure found upstream of TTS. The secondary structure was plotted with RNAalifold, and it's 5' end is denoted by a small dot at the end of the line. The color coding is at the top right; whereas darker colors show compatible pairs, and the number of types of pairs shows how many types are found at this position. Dark red colors indicate base pairs that are compatible and conserved.

parameter and align and fold the sequences into secondary structures using `RNAalifold` of the `ViennaRNA package` (Bernhart et al., 2008; Lorenz et al., 2011b). `Graphclust` was used with default parameters and additionally a window size of 110 (such that our sequences fit in one window to avoid duplicated sequences), a bitscore of 15 for the results of `cmscan`, an upper threshold of 50 clusters and 20 top sequences in each alignment for the visualization.

`Graphclust` provides covariance models (CMs) for all resulting structures (see also Subsection 5.4.4). These CMs were used to scan the remaining sequences upstream of TTS and additionally on all the transcripts in order to get a background model. Using this approach we found hairpin structures upstream of 503 of the TTS (up to 10 nucleotides distance) (Figure 68). However, no further significant secondary structures specific to occur close to TTS were found. Thus, in some cases secondary structures are present that might influence transcription termination which should be investigated in more detail in further research.

8.4.3 Experimental confirmation of selected termination signals

We selected four TTS identified with our new IE-PC approach in intergenic and coding regions to test their termination activity with an in vivo reporter gene assay. All tested TTS caused the transcription to terminate, however results show that the tested TTS with a T stretch as sequence motif terminated transcription more efficiently than the two tested TTS with AGATC motif. The fourth tested TTS includes the detected hairpin motif where 57% of the transcripts are terminated at the structural motif. For more details on the experimental verification, see Berkemer et al. (2020) and corresponding supplemental data.

8.4.4 3' UTR length for primary TTS and identification of unannotated genes

To determine 3' UTR lengths, the distance between the primary TTS and the 3' end of the preceding annotated gene was determined. The primary TTS had a median 3' UTR length of 97 nucleotides and the median 3' UTR length for all TTS is 189nt (Figure 69). To confirm that the 3' UTR regions were part of the same transcripts as the upstream coding region ends, we wanted to obtain RNAseq data to investigate whether the 3' UTR regions were part of the same transcripts as the upstream coding regions. To that end, we performed next-generation sequencing of a cDNA library generated from total RNA for RNAseq. An average of 42 million reads were obtained for three independent cDNA libraries, and with these data, we confirmed that the 3' UTR regions were part of the same transcripts as the upstream coding regions, since we found continuous RNAseq reads over the complete putative 3' UTRs. We confirmed 3' UTR regions by checking if coding region and 3' UTR are completely covered by RNAseq reads. If so, this would be indicative of uninterrupted transcription until the assigned TTS. Out of 2,631 3' UTR sequences, 2,518 show a continuous coverage within RNAseq reads (95.7%). In order to get an impression of continuously covered sequences throughout intergenic regions in the genome, we create a randomized set of sequences with similar length distribution as our set of 3' UTR sequences. Randomized 3' UTR sequences were created using `bedtools shuffle` (Quinlan and I. M. Hall, 2010). This results in 75.9% of the randomized sequences being continuously covered by RNAseq reads.

Figure 69 shows that there were also several high values for 3' UTRs with lengths of 1 kb and longer. A possible explanation for this observation is that there may be some as-yet unannotated genes between the next upstream gene end and the TTS. We performed a detailed analysis of these UTR sequences, revealing that in 356 cases a transcription start site is located in the 3' UTR, between the 3' end of the annotated gene and the TTS. Thus, 18 potential new genes are located in these 3' UTRs defined by the presence of a TSS and a TTS for the +TEX sample. The same analysis for the -TEX TTS resulted in 356 TSS-TTS pairs and thus 356 potential new genes.

8.4.5 Interaction of identified sRNAs and long 3' UTRs

The observed long 3' UTRs would allow for interactions with regulatory molecules such as proteins and RNA. Small RNAs that can potentially act as regulatory RNAs have been detected in *Haloferax* (R. Heyer et al., 2012; Straub et al., 2009) , and we analyzed whether they have the potential to interact with 3' UTRs. Twenty-one previously detected sRNAs (R. Heyer et al., 2012) were investigated for potential interactions with long 3' UTR regions using `RNAplex`, version 2.4.9 (Tafer and Hofacker, 2008) (`Vienna RNApackage 2.0`, Lorenz et al. (2011b)). `RNAplex` allows users to check for interactions of short RNA sequences with longer target sequences, and we first analyzed the 3' UTRs belonging to the TTS identified from the +TEX data set. In addition, we used the 2,631 3' UTR sequences discovered with the -TEX data set as target sequences. If several UTRs were detected for one gene, we used the longest UTR for the analysis. `RNAplex` was applied to each sRNA sequence against the long 3' UTR sequences. To have comparable values, we additionally applied `RNAplex` to each sRNA against a set of random sequences with a

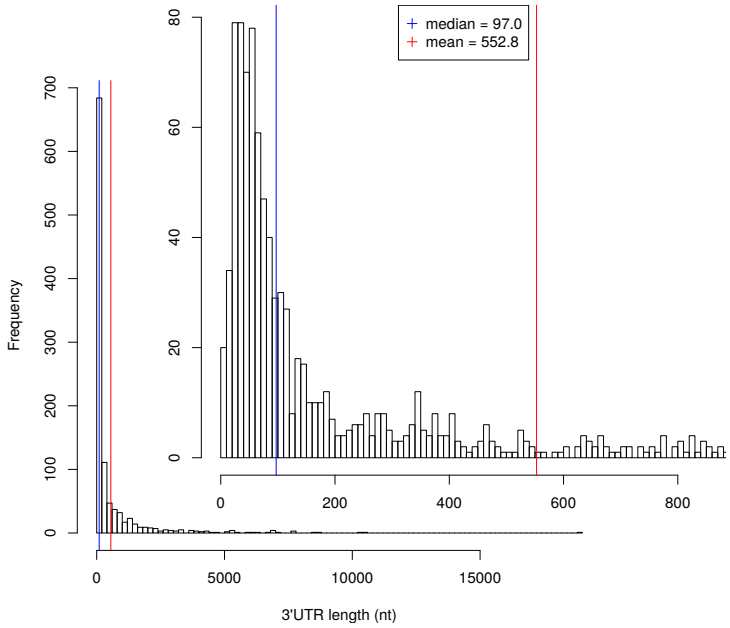


Figure 69: Histogram of 3' UTR lengths. The 3' UTR lengths for all primary TTS as calculated from the IE-PC data. Without any restriction on the length of the 3' UTR (left), we found a high number of 3' UTRs with lengths below 1,000 nucleotides and very few 3' UTRs that were longer than 1,000 nucleotides. The histogram in the inset (right) shows the frequency of 3' UTR lengths for the region up to 800 nucleotides in length.

similar length distribution as the set of 3' UTRs. Only sRNA interactions with 3' UTRs that showed more favorable binding energies than those with random sequences were taken into account. While we did not find any interactions of sRNA with the +TEX derived 3' UTRs we found clear signals for potential interactions of sRNAs with 3' UTR sequences of the -TEX data set: six different sRNAs and nine different interactions.

8.5 Concluding Remarks

The new IE-PC approach allows comprehensive, genome-wide identification of transcription termination sites. Using the new IE-PC algorithm, we were able to determine the TTS genome-wide for the archaeon *H. volcanii*. Altogether, 1,543 TTS were found with our new algorithm, representing the first unbiased, truly genome-wide approach. The dRNAseq method (+TEX) was originally established for the identification of TSS but the enrichment of primary transcripts also allows enrichment of original 3' ends as shown here. The number of TTS identified with the dRNAseq approach is clearly lower than the one identified with -TEX but is enriched for transcription termination sites and depleted for processing sites. Additional TTS might be present in the -TEX data set but more experiments are required to differentiate between the TTS and the PS in the -TEX data

set.

Among all TTS, two general sequence motifs for termination were identified, each being used in about half of the termination events. One was specific for intergenic regions, and the other one for coding regions. Both types of termination motifs (T stretch and AGATC) were confirmed using in vivo assays. Motifs in intergenic regions consisted of a stretch of Ts, while termination occurred at a C residue. This motif is similar to those found by in vitro studies in other archaea (Santangelo and Reeve, 2006; Santangelo et al., 2009; Spitalny and Thomm, 2008; Thomm et al., 1993; J. W. Brown et al., 1989) and for *S. solfataricus* and *M. mazei* using DSM (Dar et al., 2016a). The presence of TTS in coding regions has previously been observed in archaea (Dar et al., 2016a; Koide et al., 2009) and the prevailing motif we have found in coding regions was AGATC, located downstream of the termination site. The DNA sequence downstream of the TTS has been shown to influence termination efficiency in bacteria and eukaryotes (Spitalny and Thomm, 2008). Furthermore, it has been reported that the archaeal RNA polymerase interacts with downstream duplex DNA (J. W. Brown et al., 1989). Thus, it is entirely possible that a downstream termination motif exists.

With our genome-wide TTS determination approach, we can present the first comprehensive determination of 3' UTR lengths in *H. volcanii*. The median 3' UTR length for primary TTS was 97 nucleotides, which is longer than the median 3' UTR length recently reported for two archaea (with 55 nucleotides for *S. solfataricus* and 85 nucleotides for *M. mazei*) and the bacterium *B. subtilis* (40 nucleotides) (Dar et al., 2016a). The presence of transcripts with long UTRs was confirmed by RNAseq data. One explanation for these long UTRs could be that as yet undetected genes are located between the 3' ends of the upstream genes and the TTS. In particular, non-coding genes are very difficult to identify, and might not yet have been detected. Our detailed analysis of 3' UTR sequences indeed found 356 potential new genes between the 3' end of an annotated gene and a TTS. These new genes could encode non-coding RNAs. Further biological studies are required to confirm this hypothesis.

If a gene is located on the opposite strand of a TTS, an antisense RNA against this oppositely encoded gene is generated. Of the 1,543 TTS detected, 14.4% (222 TTS) of them were located in a gene on the opposite strand, thus resulting in antisense RNAs. Up to 75% of all genes were found to be associated with antisense RNAs in bacteria (Georg and Hess, 2011; Amman et al., 2018; Thomason and Storz, 2010). Studies with archaea painted a similar picture (Cohen et al., 2016; Babski et al., 2016).

The published TSS data revealed that the transcription start site for a gene can be located in an upstream gene in an operon (Babski et al., 2016). Similarly, the TTS data reported here show that transcription can be terminated in a downstream gene in an operon. Together, this allows fine-tuning of the expression of individual genes in an operon and thus challenges the simplistic view that all genes of an operon are co-transcribed under all conditions. Nucleosome coverage for Haloferax was determined previously (Ammar et al., 2012), and together with our TTS data, we showed that nucleosome coverage at termination sites was strongly reduced. For more details on operon detection and analysis of nucleosome coverage as well as further details of the study, see Berkemer et al. (2020) and corresponding supplemental files.

Part IV

Conclusion & Future Work

CHAPTER 9

Dynamic Programming in Theory and Applications

Contents

The thesis describes theoretical backgrounds and applications of dynamic programming (DP) in bioinformatics. The first and theoretical part is based on the generalization of input structures to DP algorithms whereas the second part describes applications with extensions and adaptations of existing DP algorithms to biological data.

Chapter 3 and Chapter 4 give insights into the development of DP algorithms based on algebraic dynamic programming (ADP). The extension to general data structures is an important step for further research and is not yet fully understood. While Chapter 3 describes the development of DP algorithms on tree and forest structures within the ADP framework, Chapter 4 focuses on the structure of possible input structures to alignment algorithms and further generalizes the concept to graphs, partial orders and general structures. The second part of this thesis gives an overview of possible applications of DP algorithms to biological data. Distinct input data structures as well as special applications require extended and adapted versions of DP algorithms. There are obviously many more applications and algorithms for the analysis of biological data in bioinformatics research. Projects consist of a mixture of self-implemented and/or designed algorithms together with a set of already existing algorithms. Many popular and widely-used algorithms in bioinformatics include DP techniques. All of the projects described in Chapter 6, Chapter 7 and Chapter 8 use (multiple) sequence alignments (MSAs) and covariance models (CMs) or hidden Markov models (HMMs), thus programs such as `Infernal` or `HMMer`. Structure prediction algorithms are applied in Chapter 6 in order to detect homologous tRNA and Y RNA sequences and in Chapter 8 where the data is scanned for common sequence and secondary structure motifs around the TTS. In this thesis, two modified versions of DP algorithms have been developed and implemented. Chapter 7 relies on tree structures to explore the phylogenetic relationships between archaea and bacteria using a modified Fitch algorithm to count minimal splits. Chapter 6 makes use of a modified well-known DP algorithm, too, when creating duplication alignments in order to count evolutionary duplication events among gene clusters in primates.

9.1 Theoretical Aspects of Dynamic Programming Algorithms

Chapter 3 shows that it is possible to extend DP algorithms to trees and forests as input structures and describes formal grammars for alignment and editing on tree and forest structures. The grammars only slightly differ even though the algorithms can be clearly distinguished by their output. While alignment algorithms create a smallest common superstructure of the input structures, editing algorithms output the largest common substructure. Using ADP, it is possible to independently develop and edit individual building blocks of DP algorithms such that overlapping parts can easily be reused. The difference between the algorithmic descriptions in ADP also lies in the underlying index structures, which is the preorder on trees for tree alignment and the postorder for tree editing. This shows that the definition of the index structure is essential to set the way of how the input is traversed. Therefore, a set of decomposition operators on tree and forests structures has been defined in Chapter 3 that decompose the input stepwise in an order-preserving way. Chapter 3 also includes various examples for DP algorithms on trees and forests described by formal grammars including a description of the Fitch algorithm. A modified version of this algorithm is used later in Chapter 7. The description

of DP algorithms by formal grammars gives a further advantage: In order to calculate probabilities, one can rewrite the formal grammar into its outside version and apply the inside-outside algorithm. For the alignment of two forests, probabilities for the direct match of pairs of nodes of the input are shown that result from the application of the inside-outside algorithm on the input forests.

Chapter 4 gives definitions for alignments on totally ordered sets such as strings. Based on the definition of alignment graphs, this notion is generalized to alignments of partially ordered sets and further extended to general structures. Examples for the alignment of trees and graphs are given. In the case of graph alignments, alignment columns corresponding to (mis)matches form common induced subgraphs. The pairwise alignment problem for two input graphs G_1 and G_2 therefore boils down to the problem of finding a maximum common induced subgraph (MCIS). The MCIS is a well-known NP-hard problem which can be reduced to clique finding (Barrow and Burstall, 1976). Nevertheless it is of substantial practical importance, in particular in chemoinformatics, since molecules are conveniently represented as graphs. A variety of practically applicable algorithms are therefore available (Raymond and Willett, 2002; Ehrlich and Rarey, 2011; Duesbury et al., 2018). In addition to clique-finding, DP algorithms have been explored in particular for restricted classes of graphs (Akutsu, 1993; Fomin et al., 2011). In the setting of graph alignments, it may be interesting not only to score the matches, i.e., the common induced subgraph, but also the insertions and deletions, possibly requiring modified algorithmic approaches. In the context of poset alignments, notions of alignments were explored that require less stringent conditions than the exact recovery of the structure of each input row: it also seems to be of interest to require only that the restriction to a row is an extension of the input order. In the case of graphs, a similarly relaxed condition would only require that the input is a subgraph of the restriction. RNA structures may be considered as totally ordered sets that in addition carry a graph structure defined by the base pairs. Structure annotated alignments, then, have to recover the sequence order upon restriction to the input order, while the restriction of consensus base pairing on the alignment columns only needs to be a subgraph of the input base pairings.

9.2 Application of Dynamic Programming Algorithms in Bioinformatics

The `SMORE` pipeline presented in Chapter 6 sets the state for large-scale quantitative investigations into evolution of multi-copy gene families. In particular, it provides the data required to estimate gain and loss rates and the relative effects of e.g. unequal crossover (which governs local gain and loss), retroposition (leading to insertions at novel loci), and pseudogenization (leading to loss of function and subsequent gradual disappearance of the element under consideration). This quantitative view is in particular of importance for even larger families of repetitive elements. The approach assumes perfect conservation of gene order in the vicinity to the elements of interest. While this is a very good approximation at smaller evolutionary scales, say among primate genomes, there are noticeable violations at larger scales. Fewer synteny anchors are available for more distant genomes because large fractions of the genome are diverged beyond the limits of reliable alignments. As a consequence, anchors are on average separated by larger genomic distances and thus

more likely to be separated by genome rearrangements.

Molecular phylogenetics and comparative genomics form the basis of understanding evolutionary relationships. In addition, these techniques allow inference into the most ancient types of life on Earth (e.g. Mirkin et al. (2003), Glansdorff et al. (2008), and Fournier et al. (2015)). Chapter 7 presents several existing ways of reconstructing evolutionary history based on sets of orthologous genes for archaeal and bacterial proteins. Despite an agreed upon definition of orthology, methods of creating both, sets of orthologous genes and for the reconstruction of phylogenetic trees differ significantly. The comparative analysis in Chapter 7 also includes a permutation analysis. The comparison clearly shows that the probability of seeing a single split topology in trees with relatively small numbers of genes increases significantly. Thus, the analysis suggests that groups of proteins with a higher number of genes in all domains provide more reliable data regarding their evolutionary history. Additionally, it will be beneficial to use interdisciplinary methods, e.g. including structural data to better estimate phylogenetic distances. Geochemical data can give further clues about the environmental conditions on early Earth, allowing for phylogenetic-geochemical calibrations to be made (e.g. Wolfe and Fournier (2018) and Shih et al. (2017)).

Transcription termination is not yet well understood even though it is assumed to have strong implications on regulation of transcription and translation of genes. Taken together, Chapter 8 showed that the new IE-PC approach is well-suited to identifying the complete set of TTS for a genome without any a priori limitations on the search space due to genome annotation. In comparison to the already existing approach which is based on genomic annotation (DSM), the genome-wide approach (IE-PC) leads to the detection of further TTS as well as more refined genomic locations. The T stretch termination motif detected 30 years ago was confirmed, but further additional primary and secondary structure motifs for termination were as well identified. The presence of multiple 3' UTRs for a gene provides a platform for regulatory mechanisms similar to those described in eukaryotic systems.

9.3 Discussion & Future Work

Using ADP for the development of DP algorithms has been a significant step towards further explorations of properties of DP algorithms and the extension to generalized input structures. There exist frameworks implementing ADP-style DP algorithms such as `Bellman's GAP` (Sauthoff et al., 2013) and `ADPfusion` (Höner zu Siederdissen, 2012) which allow to independently develop recursion equations, scoring algebra and memoization structures.

Limitations to DP algorithms and the ADP framework have been formulated recently in Miklós (2019). Next to the problem of sparsification and more efficient implementations of DP algorithms, the extension to generalized input structures and formulations of DP problems is an important task. Formulations of DP algorithms on trees and forests provide examples of further data structures (Chapter 3) and the detailed analysis of alignment on generalized data structures given in Chapter 4 shows that DP algorithms still bear various possible extensions. It will certainly be interesting to study such relaxed requirements

on structure preservation more systematically in future work. The fact that (multiple) alignments can be defined for very general structures, in essence for finite spaces with reasonably well-behaved notions of subspaces, suggests that alignments may be of interest as mathematical objects also for infinite spaces.

However, formal grammars seem to be only one step in further generalized formulations of DP algorithms. An extension of such tools to an easily applicable and user-friendly framework is desirable for applications in bioinformatics analysis. Possible extensions would be approaches to translate between formal grammars and easier, more intuitive problem formulations usable by non-experts. Here, further investigations could be done on the relations of weighted grammars with weighted finite automata and make use of the extensive literature and theoretical background of automata theory to gain insights into the development of DP algorithms and agree on common definitions and usage of formal grammars. Specific questions occur when dealing with the description of algorithms by regular and context-free grammars. Expressive power of formal grammars changes based on the weight structures which increases the number of possibilities for the description of DP algorithms by weighted formal grammars.

The definition of dynamic programming is based on the lookup of already calculated subsolutions, thus a specified order on the elements of the input is an essential requirement (Tendeau, 1998). Given the definition of basic data structures, the order of elements is similarly essential. Therefore, the concrete structure of the input is not important for the DP algorithm as long as every element has a clear predecessor and successor, except the first (initial) and last (terminal) elements. This is well defined for strings as their elements are a total order. Trees and forests are uniquely defined by the combination of preorder and postorder on their nodes (see Subsection 2.2.1) (Gärtner and Stadler, 2019). Thus, each element (except the first element) has a predecessor in at least one of the partial orders. Here, the question arises if index structures should be implicitly or explicitly defined as part of the ADP framework. This leads to the question of how to correctly concatenate the index structure with the building blocks of the DP algorithm, thus the grammar, scoring algebra and memoization.

This thesis contributes to the generalization of input structures, however, Chapter 4 shows limitations of the application of DP algorithms for the calculation of graph alignments which could be calculated more efficiently using algorithms to compute MCIS. Here, development of more efficient DP algorithms will be beneficial including the exploration of aspects of scoring graph alignments and the computation of pairwise and multiple alignments of graphs.

An interesting fact is the relation of DP algorithms to divide-and-conquer algorithms and greedy algorithms (Cormen et al., 2009). Divide-and-conquer algorithms recursively divide the input into disjoint substructures until the instances are small enough to reach a solution. In this way, the size of the search space is reduced and an optimal solution can be found in the case of solving an optimization problem. As the substructures are not overlapping, the algorithm can only be applied if such a division of the input into meaningful base cases is possible and leads to the optimal solution of the problem (Cormen et al., 2009). In contrast, greedy algorithms solve the optimization problem on the complete input. In each step, the algorithm chooses the currently best solution and recurses on the remaining input. DP algorithms greedily choose the best solution in each step, too, however, the currently optimal solution depends on the previous step

and cannot be derived from the global structure. Greedy algorithms can be applied if the input matches the structure of a greedoid or matroid (Korte et al., 1991; Björner and Ziegler, 1992), however, the inverse is not true as there exist algorithms without such an underlying structure that can still be solved by a greedy algorithm, such as the fractional knapsack problem (Cormen et al., 2009). Graph search algorithms are a further interesting case as breadth-first-search (BFS) can be solved by a greedy algorithm, depth-first-search (DFS), however, cannot be solved by a greedy algorithm but by a DP approach. Hence, a further goal is to find a way of defining which algorithms are able to be computed using dynamic programming and possibly find structural properties on the input similar to the greedoid structure. Characteristics of DP algorithms are known, however, no complete definition exists that can tell if a certain problem can be solved by a DP algorithm.

The applications in the second part of this thesis show that modifications of existing algorithms as well as their pure application occur frequently in various areas of bioinformatics research. Despite the wide range of applications and data sets, DP algorithms are a popular method for bioinformatics analyses. Here, a major challenge is the adaptation of existing DP algorithms to general data structures and applicability to given data sets.

The three quite distinct projects presented in the second part show that there exist various challenges when applying DP algorithms - but also algorithms in general - to biological data. Especially the size and complexity of data sets increase quickly and require adaptations of the algorithms to new constraints. Hence, clear and extendable frameworks and programs are desirable. However, requirements by the input might lead to the development of other algorithmic techniques, depending on their purpose and goals as DP algorithms are based on the memoization of intermediate results. Irrespective of the kind of input data, a further goal would be to automatize the identification of input structures and adaptation of the DP algorithms, thus a modular framework consisting of algorithmic building blocks that can be put together based on requirements of input and output data.

The results of Chapter 4 raise the question of resource efficient calculations as the example of alignments shows that solutions based on common induced subgraphs might result in more efficient programs. This is clearly as well dependent on the input data which occurs in many distinct sizes and shapes. Already bioinformatics applications require various kinds of input structures and sizes. Chapter 6, Chapter 7 and Chapter 8 gave an overview of possible applications and algorithm requirements. However, such data sets are imaginable to exists in further disciplines such as linguistics, computer science, medicine, chemistry and physics. Thus, one might want to apply such algorithms in interdisciplinary projects as data sets can be of very similar format and structure despite their very distinct contents. There exist various methods that can be applied outside of the borders of the own discipline such as alignments of words or parse trees of natural languages instead of biological sequences and secondary structures. Therefore, a future goal would be to do interdisciplinary projects and not only generalize DP algorithms towards bioinformatics data sets and applications.

Appendices

APPENDIX A

Dynamic Programming on Trees and Forests

A.1 Affine Gap Costs for Tree and Forest Alignment

A.1.1 Original grammar for affine gap costs

The following grammer expresses the seven rules for affine gap costs in forests (Schirmer, 2011; Schirmer and Giegerich, 2011). Here, different modes of scoring are applied: no-gap mode, parent-gap mode and sibling-gap mode. Parent and sibling mode indicate that the preceding node (either parent or sibling node) was considered a deletion. Correspondingly, the non-terminal symbol F denotes a no-gap state, P denotes a parent gap, and G denotes a sibling gap. This means that in P mode a gap was introduced in a node further toward the root, while in G mode a gap was introduced in a sibling. In both modes, an unbroken chain of deletions then follows on that tape.
This grammar supports different scoring functions for parent and sibling gaps. Gap opening and gap extension can be distinguished explicitly by including the two additional rules given in Equation A.2). They are useful in particular to produce a more expressive output in the backtracking step.

$$
\begin{array}{lcl}
\binom{F}{F} & \to & \binom{T}{T} \circ \binom{F}{F} \mid \binom{T}{Z} \circ \binom{F}{G} \mid \binom{Z}{T} \circ \binom{G}{F} \mid \binom{\$}{\$} \\
\binom{P}{F} & \to & \binom{T}{T} \circ \binom{P}{F} \mid \binom{T}{Z} \circ \binom{P}{G} \mid \left(\begin{smallmatrix} \widetilde{Z} \\ T \end{smallmatrix}\right) \circ \binom{P}{F} \mid \binom{\$}{\$} \\
\binom{F}{P} & \to & \binom{T}{T} \circ \binom{F}{P} \mid \left(\begin{smallmatrix} T \\ \widetilde{Z} \end{smallmatrix}\right) \circ \binom{F}{P} \mid \binom{Z}{T} \circ \binom{G}{P} \mid \binom{\$}{\$} \\
\binom{G}{F} & \to & \binom{T}{T} \circ \binom{F}{F} \mid \binom{T}{Z} \circ \binom{P}{G} \mid \left(\begin{smallmatrix} \widetilde{Z} \\ T \end{smallmatrix}\right) \circ \binom{G}{F} \mid \binom{\$}{\$} \\
\binom{F}{G} & \to & \binom{T}{T} \circ \binom{F}{F} \mid \left(\begin{smallmatrix} T \\ \widetilde{Z} \end{smallmatrix}\right) \circ \binom{F}{G} \mid \binom{Z}{T} \circ \binom{G}{P} \mid \binom{\$}{\$} \\
\binom{P}{G} & \to & \binom{T}{T} \circ \binom{P}{F} \mid \left(\begin{smallmatrix} T \\ \widetilde{Z} \end{smallmatrix}\right) \circ \binom{P}{G} \mid \left(\begin{smallmatrix} \widetilde{Z} \\ T \end{smallmatrix}\right) \circ \binom{P}{F} \\
\binom{G}{P} & \to & \binom{T}{T} \circ \binom{F}{P} \mid \left(\begin{smallmatrix} T \\ \widetilde{Z} \end{smallmatrix}\right) \circ \binom{F}{P} \mid \left(\begin{smallmatrix} \widetilde{Z} \\ T \end{smallmatrix}\right) \circ \binom{G}{P} \\
\binom{T}{T} & \to & \binom{n}{n}{}^{\downarrow}\binom{F}{F} \\
\binom{T}{Z} & \to & \binom{n}{-}{}^{\downarrow}\binom{F}{P} \\
\binom{Z}{T} & \to & \binom{-}{n}{}^{\downarrow}\binom{P}{F}
\end{array}
\tag{A.1}
$$

$$
\begin{array}{lcl}
\left(\begin{smallmatrix} T \\ \widetilde{Z} \end{smallmatrix}\right) & \to & \binom{n}{\cdot}{}^{\downarrow}\binom{F}{P} \\
\left(\begin{smallmatrix} \widetilde{Z} \\ T \end{smallmatrix}\right) & \to & \binom{\cdot}{n}{}^{\downarrow}\binom{P}{F}
\end{array}
\tag{A.2}
$$

A.1.2 Intermediary version of grammar for affine gap costs

As, in most applications, there is little reason to distinguish the parent and sibling mode gaps in the scoring function. Omitting also the explicit rules for gap extension, the grammar can be simplified considerably, see also Schirmer (2011) and Schirmer and Giegerich (2011). Here, $\binom{F}{F}$ denotes the non-gap mode, whereas the gap-mode is represented by mixed terms. In particular, $\binom{T}{Z}$ and $\binom{Z}{T}$ open gaps, while the remaining mixed terms refer to gap extensions.

The rules for $\binom{F}{F}$, $\binom{G}{F}$, and $\binom{F}{G}$ produce the same cases on their right-hand sides. The difference are the l.h.s. cases, which distinguish between no-gap mode and gap mode, thus between affine extension cost and gap opening cost. Additionally, the rules expressing parent gap modes $\binom{P}{F}$ and $\binom{F}{P}$ are recursively calling themselves. Some rules of the grammar can be condensed as they model equivalent cases. The fully compressed version is described in Subsection 3.4.2.

$$
\begin{array}{lcl}
\binom{F}{F} & \to & \binom{T}{T} \circ \binom{F}{F} \mid \binom{T}{Z} \circ \binom{F}{G} \mid \binom{Z}{T} \circ \binom{G}{F} \mid \binom{\$}{\$} \\
\binom{P}{F} & \to & \binom{T}{T} \circ \binom{P}{F} \mid \binom{T}{Z} \circ \binom{P}{F} \mid \binom{Z}{T} \circ \binom{P}{F} \mid \binom{\$}{\$} \\
\binom{F}{P} & \to & \binom{T}{T} \circ \binom{F}{P} \mid \binom{T}{Z} \circ \binom{F}{P} \mid \binom{Z}{T} \circ \binom{F}{P} \mid \binom{\$}{\$} \\
\binom{G}{F} & \to & \binom{T}{T} \circ \binom{F}{F} \mid \binom{T}{Z} \circ \binom{F}{G} \mid \binom{Z}{T} \circ \binom{G}{F} \mid \binom{\$}{\$} \\
\binom{F}{G} & \to & \binom{T}{T} \circ \binom{F}{F} \mid \binom{T}{Z} \circ \binom{F}{G} \mid \binom{Z}{T} \circ \binom{G}{F} \mid \binom{\$}{\$} \\
\binom{T}{T} & \to & \binom{n}{n} \downarrow \binom{F}{F} \\
\binom{T}{Z} & \to & \binom{n}{-} \downarrow \binom{F}{P} \\
\binom{Z}{T} & \to & \binom{-}{n} \downarrow \binom{P}{F}
\end{array}
\tag{A.3}
$$

A.2 Inside-Outside for Alignment and Editing

Inside grammars can be used to calculate two kinds of results. As we have seen above, a globally optimal solution for optimization problems, say the alignment distance between two trees, can be obtained. Alternatively, the partition function $\mathcal{Z} = \sum_\omega e^{s(\omega)/T}$ can be computed. Here, the sum runs over all configurations ω, $s(\omega)$ is the score of ω and T is a scaling parameter. For $T \to 0$, $\mathcal{Z}$ just counts the number of optimal solutions, for $T \to \infty$, all conformations are treated equally. The partition function $\mathcal{Z}$ thus provides access to a probabilistic model. This view plays a key role in practical applications. The inside or forward part calculates the probability for a possible (sub)solution whereas the outside part calculates all possible solutions while keeping one (sub)solution fixed. In this way, it is possible to calculate the overall probability of, e.g., two nodes being aligned to each other in an alignment of two trees. To this end, one calculates the partition function $\mathcal{Z}'$ that gives a value for two nodes being matched and the partition function $\mathcal{Z}$ for all possible cases of the complete alignment where those two nodes match. The desired probability is then the ratio $\mathcal{Z}'/\mathcal{Z}$ of the constrained and the unconstrained partition functions.

APPENDIX B

Dynamic Programming on Phylogenetic Trees: Towards the Last Common Ancestor

B.1 Additional Tables

COG	s	arc	bac	cat	$\overline{D}$	annotation
COG0048	1	82	625	J	0.15	Ribosomal protein S12
COG1846	74	111	2349	K	0.97	DNA-bind. transcript. regulator, MarR fam.
COG0052	1	83	605	J	0.32	Ribosomal protein S2
COG0184	4	81	626	G	0.56	Enolase
COG0522	1	83	637	J	0.33	Ribosomal protein S4 or related protein
COG0636	10	56	198	C	0.82	FoF1-type ATP synthase
COG1229	2	29	32	C	0.63	Formylmethanofuran dehydrogenase subunit A
COG1740	3	13	181	C	0.78	Ni,Fe-hydrogenase I small subunit
COG0006	15	87	838	E	0.93	Xaa-Pro aminopeptidase
COG0037	4	86	667	J	0.59	tRNA(Ile)-lysidine synthase TilS/MesJ
COG0073	5	35	605	J	0.36	tRNA-binding EMAP/Myf domain
COG0112	5	80	650	E	0.74	Glycine/serine hydroxymethyltransferase
COG0113	4	69	523	H	0.83	Delta-aminolevulinic acid dehydratase
COG0468	4	94	616	L	0.41	RecA/RadA recombinase
COG0526	32	83	1554	O	0.91	Thiol-disulfide isomerase or thioredoxin

Table 8: Data for COGs listed as examples or shown as phylogenetic trees in the main text and supplement. s:number of splits, cat:functional category.

COG	arc	bac	Cat	$\overline{D}$	annotation
COG0012*	85	617	J	0.38	Ribosome-binding ATPase YchF, GTP1/OBG family
COG0051*	84	621	J	0.41	Ribosomal protein S10
COG0080*	83	632	J	0.48	Ribosomal protein L11
COG0081*	83	626	J	0.45	Ribosomal protein L1
COG0090*	83	624	J	0.46	Ribosomal protein L2
COG0093*	83	625	J	0.36	Ribosomal protein L14
COG0094*	83	625	J	0.39	Ribosomal protein L5
COG0096*	83	625	J	0.47	Ribosomal protein S8
COG0098*	83	626	J	0.46	Ribosomal protein S5
COG0100*	83	620	J	0.34	Ribosomal protein S11
COG0103*	83	626	J	0.46	Ribosomal protein S9
COG0150	72	568	F	0.68	Phosphoribosylaminoimidazole (AIR) synthetase
COG0315	74	437	H	0.59	Molybdenum cofactor biosynthesis enzyme
COG0466	9	534	O	0.59	ATP-dependent Lon protease, bacterial type
COG0480*	81	761	J	0.44	Translation elongation factor EF-G, a GTPase
COG0533*	42	617	J	0.54	tRNA A37 threonylcarbamoyltransferase TsaD
COG0541*	82	605	U	0.61	Signal recognition particle GTPase
COG0552*	79	586	U	0.66	Signal recognition particle GTPase
COG0565	74	281	J	0.72	tRNA C32,U32 (ribose-2'-O)-methylase TrmJ
COG0643	1	38	NT	0.75	Chemotaxis protein histidine kinase CheA
COG0709	14	239	E	0.52	Selenophosphate synthase
COG1027	7	215	E	0.68	Aspartate ammonia-lyase
COG1035	20	69	C	0.54	Coenzyme F420-reducing hydrogenase, beta subunit
COG1163	83	10	J	0.65	Ribosome-interacting GTPase 1
COG1379	27	67	R	0.71	PHP family phosphoesterase with a Zn ribbon
COG1415	37	31	S	0.71	Uncharacterized protein
COG1777	47	2	K	0.67	Predicted transcriptional regulator
COG1859	36	81	J	0.63	RNA:NAD 2'-phosphotransf., TPT1/KptA fam.
COG1883	6	146	C	0.70	Na+-transp. methylmalonyl-CoA/oxaloacetate decarboxylase
COG2032	2	187	P	0.76	Cu/Zn superoxide dismutase
COG2037	29	30	C	0.58	formyltransferase
COG2069	24	34	C	0.62	CO dehydrogenase/acetyl-CoA synthase delta subunit
COG2262	60	556	J	0.68	50S ribosomal subunit-associated GTPase HflX
COG2382	5	235	P	0.61	Enterochelin esterase or related enzyme
COG2920	5	139	P	0.59	Sulfur relay (sulfurtransferase) protein
COG3252	45	34	H	0.66	Methenyltetrahydromethanopterin cyclohydrolase
COG3376	7	72	P	0.67	High-affinity nickel permease
COG3508	3	173	Q	0.67	Homogentisate 1,2-dioxygenase
COG3885	9	53	Q	0.57	Aromatic ring-opening dioxygenase, LigB subunit
COG4021	27	15	J	0.69	tRNA(His) 5'-end guanylyltransferase
COG4277	17	125	R	0.65	Predicted DNA-binding protein
COG4732	5	36	S	0.64	Predicted membrane protein
COG4754	22	67	S	0.62	Uncharacterized protein
COG4866	9	64	S	0.70	Uncharacterized protein

Table 9: Data set of SSC^{COG} showing a single split. (*) indicates COGs that are in the set of single 3-domain split groups of Harris et al.

COG	s	arc	bac	cat	$\overline{D}$	annotation
COG0109	4	30	402	HI	0.81	Polyprenyltransf.(heme O synthase)
COG0316	6	32	556	O	0.83	Fe-S cluster assembly iron-bind. prot. IscA
COG0400	5	18	385	R	0.84	Predicted esterase
COG0412	7	14	521	Q	0.93	Dienelactone hydrolase
COG0431	9	25	585	C	0.89	NAD(P)H-dependent FMN reductase
COG0569	19	76	661	P	0.93	Trk K+ transp. sys., NAD-bind. comp.
COG0604	13	39	1317	CR	0.95	NADPH:quinone or rel. Zn-dep. oxido reduct.
COG0654	6	9	1189	HC	0.94	2-polyprenyl-6-methoxyphenol hydroxylase
COG0665	14	47	984	E	0.92	Glycine/D-amino acid oxidase (deaminating)
COG0843	8	32	495	C	0.81	Heme/copper-type cytochrome/quinol oxidase, subunit 1
COG1012	29	76	1937	C	0.98	Acyl-CoA reduct. or other NAD-dep. dehyd.
COG1018	10	31	709	C	0.91	Ferredoxin-NADP reductase
COG1064	17	42	468	G	0.90	D-arabinose 1-dehydrogenase
COG1290	6	31	350	C	0.77	Cytochrome b subunit of the bc complex
COG1622	7	34	375	C	0.87	Heme/copper-type cytochrome/quinol oxidase, subunit 2
COG1764	4	12	420	V	0.78	Organic hydroperoxide reductase OsmC/OhrA
COG1845	5	24	496	C	0.88	Heme/copper-type cytochrome/quinol oxidase, subunit 3
COG2010	4	10	838	C	0.59	Cytochrome c, mono- and diheme variants
COG2128	6	18	601	P	0.85	Alkylhydroperoxidase family enzyme
COG2132	9	21	423	DPM	0.92	Multicopper oxidase with three cupredoxin domains
COG2133	12	28	581	G	0.94	Glucose/arabinose dehydrogenase, beta-propeller fold
COG2154	8	30	356	H	0.88	Pterin-4a-carbinolamine dehydratase
COG2259	8	20	530	S	0.95	Uncharact. membrane prot. YphA, DoxX/SURF4 fam.
COG2346	2	14	350	P	0.66	Truncated hemoglobin YjbI
COG2353	1	2	548	R	0.70	Polyisoprenoid-binding periplasmic protein YceI
COG4221	12	30	1124	C	0.97	NADP-dep. 3-hydroxy acid dehydrogenase YdfG
COG1152	4	23	18	C	0.39	CO dehydrogenase/acetyl-CoA synthase alpha subunit
COG1456	5	27	42	C	0.74	CO dehydrogenase/acetyl-CoA synthase gamma subunit
COG1614	2	24	36	C	0.55	CO dehydrogenase/acetyl-CoA synthase beta subunit
COG2069	1	24	34	C	0.59	CO dehydrogenase/acetyl-CoA synthase delta subunit
COG0085	1	87	619	K	0.27	DNA-directed RNA polymerase, beta/140 kD subunit
COG0086	1	82	537	K	0.25	DNA-directed RNA polymerase, beta'/160 kD subunit
COG0231	1	82	636	J	0.24	Transl. elong. f. P(EF-P)/transl. init. f. 5A(eIF-5A)
COG1110	2	35	20	L	0.85	Reverse gyrase

Table 10: Data for COGs appearing in the following data sets: 26 oxygen related COGs, 4 CODH COGs, 4 COGs mentioned by Catchpole and Forterre (2019). s:number of splits, cat:functional category.

APPENDIX C

Unbiased Map of Transcription Termination Sites in *Haloferax volcanii*

C.1 Mapping RNAsequencing data

To determine TTS, RNA was isolated from *H. volcanii* cells (from three biological replicates) and cDNA libraries were generated to allow RNA 3' end determination. Libraries were subjected to paired-end next-generation sequencing (NGS), resulting in 22 million read pairs for each library on average. Raw reads were clipped and trimmed using cutadapt version 1.10 (Martin, 2011) based on fastqc version 0.11.4 (Andrews et al., 2010) quality control reports. Reads were then mapped with segemehl (version 0.2.0) (S. Hoffmann et al., 2009; C. Otto et al., 2014), see Table 11. We used -A 94 to require higher accuracy in order to account for prokaryote mapping instead of mapping eukaryotic genomes. Mapped reads were processed using samtools version 1.3 (Li et al., 2009). The bedtools suite (bedtools v2.26.0) (Quinlan and I. M. Hall, 2010) was used to calculate genome coverage and intersection of data sets.

C.2 Dar-Sorek-Method

Based on the Dar-Sorek-Method (DSM) (Dar et al., 2016a; Dar et al., 2016b), putative TTS were retrieved from the mapping data. We only used read pairs and at least a coverage of 4 for every valid position. For each annotated region in the genome, all inserts overlapping with the annotated region were collected. Then, for each position downstream of a gene, all collected reads that end at this position were summarised, excluding inserts longer than 500 bp. The average insert length of all the corresponding read pairs was defined as the length of the target downstream region. For all read pairs where the insert

sample	replika	mapped	uniquely mapped	unmapped	% mapped
TEX-	S1	54,339,610	40,848,267	1,966,445	95
	S2	45,672,059	34,466,376	1,638,848	95
	S3	60,258,543	45,649,822	1,638,848	97
TEX+	S1	59,993,014	46,773,468	5,737,678	89
	S2	38,462,663	29,191,284	2,931,762	91
	S3	44,117,608	34,786,898	2,825,896	92
Wildtype	S1	54,684,129	42,626,326	1,934,439	96
	S2	50,210,254	40,447,008	1,566,743	96
	S3	42,503,303	38,733,269	810,822	98

Table 11: Numbers of mapped reads, uniquely mapped reads and unmapped reads for RNASeq data. TEX+ includes enriched primary (unprocessed) 3' ends, thus transcription termination sites and sequences starting from the transcript fragments' 3' ends. TEX- is sequenced starting from the transcript fragments' 3' ends without enrichment and the wildtype is sequenced starting from the transcripts 5'ends.

overlapped an annotated region, the position with highest read-end coverage inside the target downstream region of an annotated 3' end was reported as a TTS (Table 12 and Figure 61).

C.3 Internal Enrichment

The software, Internal-Enrichment (IE), was implemented in perl and is available at Bioinformatics Leipzig (`http://www.bioinf.uni-leipzig.de/publications/supplements/18-059`). Based on the distributions, native fragment end scores can be assigned an empirical p-value, evaluating its likelihood to occur by change alone. We ran IE using all position showing signals for starts and ends (-mr 1), we omitted read fragments with length more than 100 nt (-mf 100), and the geometric mean as a method to calculate a score for the given signals (-mode GeomMean). We only included signals that were present in all three replica such that each showed a maximal empirical p-value of 0.05. The geometric mean of all empirical p-values for one position is used as the final score.

chromosome	chr. length (kb)	intergenic	coding	total
CHR	2,848	2,046	378	2,424
pHV1	85	70	22	92
pHV3	438	204	18	222
pHV4	636	353	64	417
total	-	2,673	482	3,155

Table 12: Number of TTS determined using DSM for distinct parts of the genome (main chromosome and chromosomal plasmids pHV1, pHV3 and pHV4) and for coding and intergenic regions.

List of Abbreviations

ADP algebraic dynamic programming.

BLAST basic local alignment search tool.

CFG context-free grammar.

CMs covariance models.

COGs conserved orthologous groups of proteins.

DAG directed acyclic graph.

DNA deoxyribonucleic acid.

DP dynamic programming.

ER equivalence relation.

HGT horizontal gene transfer.

HMMs hidden Markov models.

LCA lowest common ancestor.

LGT lateral gene transfer.

lncRNA long non-coding RNA.

LUCA lowest universal common ancestor.

MCIS maximum common induced subgraph.

miRNA micro RNA.

ML maximum likelihood.

MP maximum parsimony.

mRNA messenger RNA.

MSA multiple sequence alignment.

ncRNA non-coding RNA.

NGS Next Generation Sequencing.

NJ neighbor-joining.

NUTs nearly universal trees.

ORF open reading frame.

PDB protein data bank.

PO partial order.

PS processing sites.

RNA ribonucleic acid.

rRNA ribosomal RNA.

snoRNA small nucleolar RNA.

snRNA small nuclear RNA.

TO total order.

TOL tree of life.

tRNA transfer RNA.

TSS transcription start site.

TTS transcription termination site.

UTR untranslated region.

Bibliography

Adam, P. S., G. Borrel, and S. Gribaldo (2018). "Evolutionary history of carbon monoxide dehydrogenase/acetyl-CoA synthase, one of the oldest enzymatic complexes". In: *Proceedings of the National Academy of Sciences* 115.6, E1166–E1173. DOI: 10.1073/pnas.1716667115.

Afgan, E., D. Baker, B. Batut, M. Van Den Beek, D. Bouvier, M. Čech, J. Chilton, D. Clements, N. Coraor, B. A. Grüning, et al. (2018). "The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update". In: *Nucleic acids research* 46.W1, W537–W544. DOI: 10.1093/nar/gky379.

Akutsu, T. (1993). "A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree". In: *IEICE transactions on fundamentals of electronics, communications and computer sciences* 76.9, pp. 1488–1493. DOI: 10.3390/a6010119.

Alberts, B., A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter (2008). *Molecular biology of the cell: 5th edition*. Garland Science. DOI: 10.1201/9780203833445.

Altenhoff, A. M. and C. Dessimoz (2009). "Phylogenetic and functional assessment of orthologs inference projects and methods". In: *PLoS Comput Biol.* 5, e1000262. DOI: 10.1371/journal.pcbi.1000262.

Altschul, S. F., W. Gish, W. Miller, E. W. Myers, and D. J. Lipman (1990). "Basic local alignment search tool". In: *Journal of molecular biology* 215.3, pp. 403–410. DOI: 10.1006/jmbi.1990.9999.

Altschul, S. F., T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman (1997). "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs". In: *Nucleic Acids Res.* 25, pp. 3389–3402. DOI: 10.1093/nar/25.17.3389.

Amman, F., A. D'Halluin, R. Antoine, L. Huot, I. Bibova, K. Keidel, S. Slupek, P. Bouquet, L. Coutte, S. Caboche, et al. (2018). "Primary transcriptome analysis reveals importance of IS elements for the shaping of the transcriptional landscape of Bordetella pertussis". In: *RNA biology* 15.7, pp. 967–975. DOI: 10.1080/15476286.2018.1462655.

Ammar, R., D. Torti, K. Tsui, M. Gebbia, T. Durbic, G. D. Bader, G. Giaever, and C. Nislow (2012). "Chromatin is an ancient innovation conserved between Archaea and Eukarya". In: *Elife* 1, e00078. DOI: 10.7554/elife.00078.

Amstutz, H., P. Munz, W. D. Heyer, U. Leupoid, and J. Kohli (1985). "Concerted evolution of tRNA genes: intergenic conversion among three unlinked serine tRNA genes in *S. pombe*". In: *Cell* 40, pp. 879–886. DOI: 10.1016/0092-8674(85)90347-2.

Andrews, S. et al. (2010). *FastQC: a quality control tool for high throughput sequence data.*

Arimbasseri, A. G., K. Rijal, and R. J. Maraia (2013). "Transcription termination by the eukaryotic RNA polymerase III". In: *Biochimica et Biophysica Acta (BBA)-Gene Regulatory Mechanisms* 1829.3-4, pp. 318–330. DOI: 10.1126/science.1237934.

Arnold, C. and C. L. Nunn (2010). "Phylogenetic Targeting of Research Effort in Evolutionary Biology". In: *Amer. Nat.* 176, pp. 601–612. DOI: 10.1086/656490.

Arnold, C. and P. F. Stadler (2010). "Polynomial algorithms for the Maximal Pairing Problem: efficient phylogenetic targeting on arbitrary trees". In: *Alg. Mol. Biol.* 5, p. 25. DOI: 10.1186/1748-7188-5-25.

Babski, J., K. A. Haas, D. Näther-Schindler, F. Pfeiffer, K. U. Förstner, M. Hammelmann, R. Hilker, A. Becker, C. M. Sharma, A. Marchfelder, et al. (2016). "Genome-wide identification of transcriptional start sites in the haloarchaeon Haloferax volcanii based on differential RNA-Seq (dRNA-Seq)". In: *BMC genomics* 17.1, p. 629. DOI: 10.1186/s12864-016-2920-y.

Baichoo, S. and C. A. Ouzounis (2017). "Computational complexity of algorithms for sequence comparison, short-read assembly and genome alignment". In: *Biosystems* 156/157, pp. 72–85. DOI: 10.1016/j.biosystems.2017.03.003.

Bailey, T. L., M. Boden, F. A. Buske, M. Frith, C. E. Grant, L. Clementi, J. Ren, W. W. Li, and W. S. Noble (2009). "MEME SUITE: tools for motif discovery and searching". In: *Nucleic acids research* 37.suppl_2, W202–W208. DOI: 10.1093/nar/gkp335.

Barrow, H. G. and R. M. Burstall (1976). "Subgraph isomorphism, matching relational structures and maximal cliques". In: *Information Processing Letters* 4, pp. 83–84. DOI: 10.1016/0020-0190(76)90049-1.

Baumann, P., S. A. Qureshi, and S. P. Jackson (1995). "Transcription: new insights from studies on Archaea". In: *Trends in Genetics* 11.7, pp. 279–283. DOI: 10.1016/s0168-9525(00)89075-7.

Becerra, A., L. Delaye, S. Islas, and A. Lazcano (2007). "The Very Early Stages of Biological Evolution and the Nature of the Last Common Ancestor of the Three Major Cell Domains". In: *Annual Review of Ecology, Evolution, and Systematics* 38.1, pp. 361–379. DOI: 10.1146/annurev.ecolsys.38.091206.095825.

Bell, S. D. (2005). "Archaeal transcriptional regulation–variation on a bacterial theme?" In: *Trends in microbiology* 13.6, pp. 262–265. DOI: 10.1016/j.tim.2005.03.015.

Bellman, R. (1952). "On the theory of dynamic programming". In: *Proceedings of the National Academy of Sciences* 38.8, pp. 716–719.

Bellman, R. et al. (1954). "The theory of dynamic programming". In: *Bulletin of the American Mathematical Society* 60.6, pp. 503–515. DOI: 10.1090/s0002-9904-1954-09848-8.

Bennett, E. A., H. Keller, R. E. Mills, S. Schmidt, J. V. Moran, O. Weichenrieder, and S. E. Devine (2008). "Active Alu retrotransposons in the human genome". In: *Genome research* 18.12, pp. 1875–1883. DOI: 10.1101/gr.081737.108.

Bergeron, F., G. Labelle, and P. Leroux (1998). *Combinatorial species and tree-like structures*. Vol. 67. Cambridge University Press. DOI: 10.1017/cbo9781107325913.

Berkemer, S. J., L.-K. Maier, F. Amman, S. H. Bernhart, J. Wörtz, P. Märkle, F. Pfeiffer, P. F. Stadler, and A. Marchfelder (2020). "Identification of RNA 3' ends and termination sites in Haloferax volcanii". In: *RNA Biology* 0.0, pp. 1–14. DOI: 10.1080/15476286.2020.1723328.

Berkemer, S. J., R. R. Chaves, A. Fritz, M. Hellmuth, M. Hernandez-Rosales, and P. F. Stadler (2015). "Spiders can be recognized by counting their legs". In: *Mathematics in Computer Science* 9.4, pp. 437–441. DOI: 10.1007/s11786-015-0233-1.

Berkemer, S. J., A. Hoffmann, C. R. Murray, and P. F. Stadler (2017a). "SMORE: Synteny Modulator of Repetitive Elements". In: *Life* 7.4, p. 42. DOI: 10.3390/life7040042.

Berkemer, S. J., C. Höner zu Siederdissen, and P. F. Stadler (2017b). "Algebraic dynamic programming on trees". In: *Algorithms* 10, p. 135. DOI: 10.3390/a10040135.

Berkemer, S. J., C. Höner zu Siederdissen, and P. F. Stadler (2019). "Compositional Properties of Alignments". In: *Mathematics in Computer Science, submitted*.

Berkemer, S. J. and S. E. McGlynn (2020). "Phylogenetic domain separation of protein families constrains functional inference of LUCA". In: *Molecular Biology and Evolution, under review*.

Berman, H. M., J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne (2000). "The protein data bank". In: *Nucleic acids research* 28.1, pp. 235–242.

Bermúdez-Santana, C., C. Stephan-Otto Attolini, T. Kirsten, J. Engelhardt, S. J. Prohaska, S. Steigele, and P. F. Stadler (2010). "Genomic Organization of Eukaryotic tRNAs". In: *BMC Genomics* 11, p. 270. DOI: 10.1186/1471-2164-11-270.

Bernhart, S. H., I. L. Hofacker, S. Will, A. R. Gruber, and P. F. Stadler (2008). "RNAalifold: improved consensus structure prediction for RNA alignments". In: *BMC bioinformatics* 9.1, p. 474. DOI: 10.1186/1471-2105-9-474.

Bhattacharya, T. et al. (2018). "Studying Language Evolution in the Age of Big Data". In: *J. Language Evol.* 3, pp. 94–129. DOI: 10.1093/jole/lzy004.

Bille, P. (2005). "A survey on tree edit distance and related problems". In: *Theor. Comput. Sci.* 337, pp. 217–239. DOI: 10.1016/j.tcs.2004.12.030.

Bird, R. and O. De Moor (1993). "From dynamic programming to greedy algorithms". In: *Formal program development*. Springer, pp. 43–61. DOI: 10.1007/3-540-57499-9_16.

Björner, A. and G. M. Ziegler (1992). "Introduction to Greedoids". In: *Matroid Applications*. Ed. by N. White. Cambridge University Press, pp. 284–357. DOI: 10.1017/cbo9780511662041.009.

Bompfünewerer, A. F., R. Backofen, S. H. Bernhart, J. Hertel, I. L. Hofacker, P. F. Stadler, and S. Will (2008). "Variations on RNA folding and alignment: lessons from Benasque". In: *Journal of mathematical biology* 56.1-2, pp. 129–144. DOI: 10.1007/s00285-007-0107-5.

Bonizzoni, P. and G. Della Vedova (2001). "The complexity of multiple sequence alignment with SP-score that is a metric". In: *Theor. Comp. Sci.* 259, pp. 63–79. DOI: 10.1016/S0304-3975(99)00324-2.

Boria, I., A. R. Gruber, A. Tanzer, S. Bernhart, R. Lorenze, M. M. Mueller, I. L. Hofacker, and P. F. Stadler (2010). "Nematode sbRNAs: homologs of vertebrate Y RNAs". In: *J. Mol. Evol.* 70, pp. 346–358. DOI: 10.1007/s00239-010-9332-4.

Bringmann, K., P. Gawrychowski, S. Mozes, and O. Weimann (2018). "Tree Edit Distance Cannot be Computed in Strongly Subcubic Time (unless APSP can)". In: *SODA 2018*. DOI: 10.1137/1.9781611975031.77.

Brown, J. W., C. J. Daniels, J. N. Reeve, and J. Konisky (1989). "Gene Structure, Organization, And Expression In Archaebacteria". In: *CRC Critical Reviews in Microbiology* 16.4, pp. 287–337. DOI: 10.3109/10408418909105479.

Bunke, H. (1997). "On a relation between graph edit distance and maximum common subgraph". In: *Pattern Recognition Letters* 18, pp. 689–694. DOI: 10.1016/S0167-8655(97)00060-3.

Canzar, S. and S. L. Salzberg (2017). "Short read mapping: An algorithmic tour". In: *Proceedings of the IEEE* 105.3, pp. 436–458. DOI: 10.1109/jproc.2015.2455551.

Capra, J. A., M. Stolzer, D. Durand, and K. S. Pollard (2013). "How old is my gene?" In: *Trends Genet* 29, pp. 659–668. DOI: 10.1016/j.tig.2013.07.001.

Carrillo, H. and D. J. Lipman (1988). "The multiple sequence alignment problem in biology". In: *SIAM J. Appl. Math.* 48, pp. 1073–1082. DOI: 10.1137/0148063.

Case, R. J., Y. Boucher, I. Dahllöf, C. Holmström, W. F. Doolittle, and S. Kjelleberg (2007). "Use of 16S rRNA and rpoB genes as molecular markers for microbial ecology studies". In: *Appl. Environ. Microbiol.* 73.1, pp. 278–288. DOI: 10.1128/aem.01177-06.

Catchpole, R. and P. Forterre (2019). "Positively twisted: The complex evolutionary history of Reverse Gyrase suggests a non-hyperthermophilic Last Universal Common Ancestor". In: *bioRxiv*, p. 524215. DOI: 10.1101/524215.

Charlebois, R. L. and W. F. Doolittle (2004). "Computing prokaryotic gene ubiquity: rescuing the core from extinction". In: *Genome research* 14.12, pp. 2469–2477. DOI: 10.1101/gr.3024704.

Chauve, C., J. Courtiel, and Y. Ponty (2016). "Counting, generating and sampling tree alignments". In: *International Conference on Algorithms for Computational Biology*. Springer, pp. 53–64. DOI: 10.1007/978-3-319-38827-4_5.

Chen, W. (2001). "New algorithm for ordered tree-to-tree correction problem". In: *Journal of Algorithms* 40.2, pp. 135–158. DOI: 10.1006/jagm.2001.1170.

Chomsky, N. (1959). "On certain formal properties of grammars". In: *Information and control* 2.2, pp. 137–167. DOI: 10.1016/s0019-9958(59)90362-6.

Chomsky, N. and M. P. Schützenberger (1963). "The algebraic theory of context-free languages". In: *Studies in Logic and the Foundations of Mathematics* 35, pp. 118–161. DOI: 10.1016/s0049-237x(09)70104-1.

Christov, C. P., T. J. Gardiner, D. Szüts, and T. Krude (2006). "Functional Requirement of Noncoding Y RNAs for Human Chromosomal DNA Replication". In: *Mol. Cell. Biol.* 26, pp. 6993–7004. DOI: 10.1128/mcb.01060-06.

Cohen, O., S. Doron, O. Wurtzel, D. Dar, S. Edelheit, I. Karunker, E. Mick, and R. Sorek (2016). "Comparative transcriptomics across the prokaryotic tree of life". In: *Nucleic acids research* 44.W1, W46–W53. DOI: 10.1093/nar/gkw394.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to Algorithms. third edition*. DOI: 10.2307/2583667.

Costa-Silva, J., D. Domingues, and F. M. Lopes (2017). "RNA-Seq differential expression analysis: An extended review and a software tool". In: *PloS one* 12.12, e0190152. DOI: 10.1371/journal.pone.0190152.

Cysouw, M. and H. Jung (2007). "Cognate Identification and Alignment Using Practical Orthographies". In: *Proceedings of Ninth Meeting of the ACL Special Interest Group in Computational Morphology and Phonology*. Association for Computational Linguistics, pp. 109–116. URL: https://www.aclweb.org/anthology/W/W07/W07-1314.pdf.

Dalquen, D. A., A. M. Altenhoff, G. H. Gonnet, and C. Dessimoz (2013). "The Impact of Gene Duplication, Insertion, Deletion, Lateral Gene Transfer and Sequencing Error on

Orthology Inference: A Simulation Study". In: *PLoS ONE* 8, e56925. DOI: 10.1371/journal.pone.0056925.

Dar, D., D. Prasse, R. A. Schmitz, and R. Sorek (2016a). "Widespread formation of alternative 3' UTR isoforms via transcription termination in archaea". In: *Nature microbiology* 1.10, p. 16143. DOI: 10.1038/nmicrobiol.2016.143.

Dar, D., M. Shamir, J. Mellin, M. Koutero, N. Stern-Ginossar, P. Cossart, and R. Sorek (2016b). "Term-seq reveals abundant ribo-regulation of antibiotics resistance in bacteria". In: *Science* 352.6282, aad9822. DOI: 10.1126/science.aad9822.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life (1st ed.)*

De Moor, O. (1994). "Categories, relations and dynamic programming". In: *Mathematical Structures in Computer Science* 4.1, pp. 33–69. DOI: 10.1017/s0960129500000360.

Do, C. B., M. S. Mahabhashyam, M. Brudno, and S. Batzoglou (2005). "ProbCons: Probabilistic consistency-based multiple sequence alignment". In: *Genome Res.* 15, pp. 330–340. DOI: 10.1101/gr.2821705.

Dong, S. and D. B. Searls (1994). "Gene structure prediction by linguistic methods". In: *Genomics* 23.3, pp. 540–551. DOI: 10.1006/geno.1994.1541.

Droste, M. and W. Kuich (2009). "Semirings and formal power series". In: *Handbook of Weighted Automata*. Springer, pp. 3–28. DOI: 10.1007/978-3-642-01492-5_1.

Duesbury, E., J. Holliday, and P. Willett (2018). "Comparison of Maximum Common Subgraph Isomorphism Algorithms for the Alignment of 2D Chemical Structures". In: *ChemMedChem* 13, pp. 588–598. DOI: 10.1002/cmdc.201700482.

Dulucq, S. and L. Tichit (2003). "RNA secondary structure comparison: exact analysis of the Zhang-Shasha tree edit algorithm". In: *Theoretical Computer Science* 306.1, pp. 471–484. DOI: 10.1016/s0304-3975(03)00323-2.

Durbin, R., S. R. Eddy, A. Krogh, and G. Mitchison (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge, UK: Cambridge University Press. DOI: 10.1017/cbo9780511790492.002.

Eddy, S. R. (2011). "Accelerated profile HMM searches". In: *PLoS computational biology* 7.10, e1002195. DOI: 10.1371/journal.pcbi.1002195.

Eddy, S. R. (1998). "Profile hidden Markov models." In: *Bioinformatics (Oxford, England)* 14.9, pp. 755–763. DOI: 10.1093/bioinformatics/14.9.755.

Edgar, R. C. and S. Batzoglou (2006). "Multiple sequence alignment". In: *Curr Opin Struct Biol* 16, pp. 368–373. DOI: 10.1016/j.sbi.2006.04.004.

Edgar, R. C. (2004). "MUSCLE: multiple sequence alignment with high accuracy and high throughput". In: *Nucleic Acids Res.* 32, pp. 1792–1797. DOI: 10.1093/nar/gkh340.

Eger, S. (2013). "Sequence alignment with arbitrary steps and further generalizations, with applications to alignments in linguistics". In: *Information Sci.* 237, pp. 287–304. DOI: 10.1016/j.ins.2013.02.031.

Ehrlich, H.-C. and M. Rarey (2011). "Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review". In: *Wiley Interdisciplinary Reviews: Computational Molecular Science* 1, pp. 68–79. DOI: 10.1002/wcms.5.

Eigen, M., B. F. Lindemann, M. Tietze, R. Winkler-Oswatitsch, A. W. M. Dress, and A. von Haeseler (1989). "How old is the genetic code? Statistical geometry of tRNA provides an answer". In: *Science* 244, pp. 673–679. DOI: 10.1126/science.2497522.

Eigen, M. and R. Winkler-Oswatitsch (1981). "Transfer-RNA, an early gene?" In: *Naturwissenschaften* 68, pp. 282–292. DOI: 10.1007/bf01047470.

Elemento, O. and O. Gascuel (2005). "An exact and polynomial distance-based algorithm to reconstruct single copy tandem duplication trees". In: *J. Discr. Algo.* 3, pp. 362–374. DOI: 10.1016/j.jda.2004.08.013.

Elias, I. (2006). "Settling the Intractability of Multiple Alignment". In: *J. Comp. Biol.* 13, pp. 1323–1339. DOI: 10.1089/cmb.2006.13.1323.

Eme, L., A. Spang, J. Lombard, C. W. Stairs, and T. J. Ettema (2017). "Archaea and the origin of eukaryotes". In: *Nature Reviews Microbiology* 15.12, p. 711. DOI: 10.1038/nrmicro.2017.133.

Emmert-Streib, F., M. Dehmer, and Y. Shi (2016). "Fifty years of graph matching, network alignment and network comparison". In: *Information Sci.* 346/347, pp. 180–197. DOI: 10.1016/j.ins.2016.01.074.

Fagin, R., R. Kumar, and D. Sivakumar (2003). "Comparing Top-k Lists". In: *SIAM J. Discr. Math.* 17, pp. 134–160. DOI: 10.1137/S0895480102412856.

Farris, A. D., G. Koelsch, G. J. Pruijn, W. J. van Venrooij, and J. B. Harley (1999). "Conserved features of Y RNAs revealed by automated phylogenetic secondary structure analysis". In: *Nucl. Ac. Res.* 27, pp. 1070–8. DOI: 10.1093/nar/27.4.1070.

Felsenstein, J. (1999). "PHYLIP (Phylogeny Inference Package) and manual". In: URL: http://evolution.%20genetics.%20washington.%20edu/phylip.%20html.

Felsenstein, J. (1981). "Evolutionary trees from DNA sequences: a maximum likelihood approach". In: *Journal of molecular evolution* 17.6, pp. 368–376. DOI: 10.1007/bf01734359.

Feng, D.-F. and R. F. Doolittle (1987). "Progressive sequence alignment as a prerequisite to correct phylogenetic trees". In: *J. Mol. Evol.* 25, pp. 351–360. DOI: 10.1007/BF02603120.

Fitch, W. M. (1971). "Towards defining the course of evolution: minimum change for a specific tree topology". In: *Syst Zool* 20, pp. 406–416. DOI: 10.2307/2412116.

Fitch, W. M. (1970). "Distinguishing Homologous from Analogous Proteins". In: *Syst. Biol.* 19, pp. 99–113. DOI: 10.2307/2412448.

Fitch, W. M. and E. Margoliash (1967). "Construction of phylogenetic trees". In: *Science* 155.3760, pp. 279–284. DOI: 10.1126/science.155.3760.279.

Flajolet, P. and R. Sedgewick (2009). *Analytic combinatorics*. cambridge University press. DOI: 10.1017/cbo9780511801655.

Florentz, C., F. Jühling, J. Pütz, C. Sauter, P. F. Stadler, and R. Giegé (2012). "Structure of transfer RNAs: a function-driven refined view". In: *Wiley Interdiscip Rev RNA* 3, pp. 37–61. DOI: 10.1002/wrna.103.

Fomin, F. V., I. Todinca, and Y. Villanger (2011). "Exact algorithm for the maximum induced planar subgraph problem". In: *Proceedings of the 19th European conference on Algorithms*. Ed. by C. Demetrescu and M. M. Halldórsson. Vol. 6942. Lecture Notes Comp. Sci. Berlin, Heidelberg: Springer-Verlag, pp. 287–298. DOI: 10.1007/978-3-642-23719-5_25.

Forslund, K. et al. (2017). "Gearing up to handle the mosaic nature of life in the quest for orthologs". In: *Bioinformatics* 34.2. Ed. by J. Kelso, pp. 323–329. DOI: 10.1093/bioinformatics/btx542.

Forterre, P. (2006). "Three RNA cells for ribosomal lineages and three DNA viruses to replicate their genomes: A hypothesis for the origin of cellular domain". In: *Proceedings of the National Academy of Sciences* 103.10, pp. 3669–3674. DOI: 10.1073/pnas.0510333103.

Forterre, P. and S. Gribaldo (2007). "The origin of modern terrestrial life". In: *HFSP journal* 1.3, pp. 156–168. DOI: 10.2976/1.2759103.

Fouqueau, T., F. Blombach, G. Cackett, A. E. Carty, D. M. Matelska, S. Ofer, S. Pilotto, D. K. Phung, and F. Werner (2018). "The cutting edge of archaeal transcription". In: *Emerging Topics in Life Sciences* 2.4, pp. 517–533. DOI: 10.1042/etls20180014.

Fouqueau, T., F. Blombach, and F. Werner (2017). "Evolutionary origins of two-barrel RNA polymerases and site-specific transcription initiation". In: *Annual review of microbiology* 71, pp. 331–348. DOI: 10.1146/annurev-micro-091014-104145.

Fournier, G. P., C. P. Andam, and J. P. Gogarten (2015). "Ancient horizontal gene transfer and the last common ancestors". In: *BMC evolutionary biology* 15.1, p. 70. DOI: 10.1186/s12862-015-0350-0.

Frenkel, F. E., M. B. Chaley, E. V. Korotkov, and K. G. Skryabin (2004). "Evolution of tRNA-like sequences and genome variability". In: *Gene* 335, pp. 57–71. DOI: 10.1016/j.gene.2004.03.005.

Fried, C., W. Hordijk, S. J. Prohaska, C. R. Stadler, and P. F. Stadler (2004). "The Footprint Sorting Problem". In: *J. Chem. Inf. Comput. Sci.* 44, pp. 332–338. DOI: 10.1021/ci030411+.

Galperin, M. Y., K. S. Makarova, Y. I. Wolf, and E. V. Koonin (2014). "Expanded microbial genome coverage and improved protein family annotation in the COG database". In: *Nucleic acids research* 43.D1, pp. D261–D269. DOI: 10.1093/nar/gku1223.

Gärtner, F. and P. F. Stadler (2019). "Direct Superbubble Detection". In: *Algorithms* 12.4, p. 81. DOI: 10.3390/a12040081.

Geiß, M., P. F. Stadler, and M. Hellmuth (2019). "Reciprocal Best Match Graphs". In: *arXiv preprint arXiv:1903.07920*.

Georg, J. and W. R. Hess (2011). "cis-antisense RNA, another level of gene regulation in bacteria". In: *Microbiol. Mol. Biol. Rev.* 75.2, pp. 286–300. DOI: 10.1128/mmbr.00032-10.

Giegerich, R. (2000). "A systematic approach to dynamic programming in bioinformatics". In: *Bioinformatics* 16.8, pp. 665–677. DOI: 10.1093/bioinformatics/16.8.665.

Giegerich, R. and C. Meyer (2002). "Algebraic Dynamic Programming". In: *Algebraic Methodology And Software Technology*. Vol. 2422. Springer, pp. 243–257. DOI: 10.1007/3-540-45719-4_24.

Giegerich, R. and H. Touzet (2014). "Modeling Dynamic Programming Problems over Sequences and Trees with Inverse Coupled Rewrite Systems". In: *Algorithms*, pp. 62–144. DOI: 10.3390/a7010062.

Glansdorff, N., Y. Xu, and B. Labedan (2008). "The last universal common ancestor: emergence, constitution and genetic legacy of an elusive forerunner". In: *Biology direct* 3.1, p. 29. DOI: 10.1186/1745-6150-3-29.

Glantz, H. T. (1956). "On the recognition of information with a digital computer". In: *Proceedings of the 1956 11th ACM national meeting*. ACM, pp. 126–129. DOI: 10.1145/320868.320878.

Goldenfeld, N., T. Biancalani, and F. Jafarpour (2017). "Universal biology and the statistical mechanics of early life". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 375.2109, p. 20160341. DOI: 10.1098/rsta.2016.0341.

Goldman, A. D., T. M. Bernhard, E. Dolzhenko, and L. F. Landweber (2012). "LUCApedia: a database for the study of ancient life". In: *Nucleic acids research* 41.D1, pp. D1079–D1082. DOI: 10.1093/nar/gks1217.

Goodman, J. (1998). "Parsing inside-out". In: *arXiv preprint cmp-lg/9805007*.

Goodman, J. (1999). "Semiring parsing". In: *Computational Linguistics* 25.4, pp. 573–605.

Gotoh, O. (1982). "An improved algorithm for matching biological sequences". In: *J. Mol. Biol.* 162, pp. 705–708. DOI: 10.1016/0022-2836(82)90398-9.

Gotoh, O. (1986). "Alignment of three biological sequences with an efficient traceback procedure". In: *J. theor. Biol.* 121, pp. 327–337. DOI: 10.1016/S0022-5193(86)80112-6.

Grabherr, M. G., P. Russell, M. Meyer, E. Mauceli, J. Alföldi, F. Di Palma, and K. Lindblad-Toh (2010). "Genome-wide synteny through highly sensitive sequence alignment: *Satsuma*". In: *Bioinformatics* 26, pp. 1145–1151. DOI: 10.1093/bioinformatics/btq102.

Grasso, C. and C. Lee (2004). "Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems". In: *Bioinformatics* 20, pp. 1546–1556. DOI: 10.1093/bioinformatics/bth126.

Gribaldo, S., A. M. Poole, V. Daubin, P. Forterre, and C. Brochier-Armanet (2010). "The origin of eukaryotes and their relationship with the Archaea: are we at a phylogenomic impasse?" In: *Nature Reviews Microbiology* 8.10, p. 743. DOI: 10.1038/nrmicro2426.

Hahne, F. and R. Ivanek (2016). "Visualizing genomic data using Gviz and bioconductor". In: *Statistical Genomics*. Springer, pp. 335–351. DOI: 10.1007/978-1-4939-3578-9_16.

Hall, A., C. Turnbull, and T. Dalmay (2013). "Y RNAs: recent developments". In: *Biomol. Concepts* 4, pp. 103–110. DOI: 10.1515/bmc-2012-0050.

Hamming, R. W. (1950). "Error Detecting and Error Correcting Codes". In: *Bell System Technical Journal* 29.2, pp. 147–160. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

Harris, J. K., S. T. Kelley, G. B. Spiegelman, and N. R. Pace (2003). "The genetic core of the universal ancestor". In: *Genome research* 13.3, pp. 407–412. DOI: 10.1101/gr.652803.

Hartigan, J. A. (1973). "Minimum mutation fits to a given tree". In: *Biometrics* 29, pp. 53–65. DOI: 10.2307/2529676.

Hasse, H. (1985). *Über die Klassenzahl abelscher Zahlkörper*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-69886-6.

Hellmuth, M., M. Hernández Rosales, K. T. Huber, V. Moulton, P. F. Stadler, and N. Wieseke (2013). "Orthology relations, symbolic ultrametrics, and cographs". In: *Journal of mathematical biology* 66.1-2, pp. 399–420. DOI: 10.1007/s00285-012-0525-x.

Hellmuth, M. and N. Wieseke (2015). "On symbolic ultrametrics, cotree representations, and cograph edge decompositions and partitions". In: *International Computing and Combinatorics Conference*. Springer, pp. 609–623. DOI: 10.1007/978-3-319-21398-9_48.

Hellmuth, M., N. Wieseke, M. Lechner, H.-P. Lenhof, M. Middendorf, and P. F. Stadler (2015). "Phylogenomics with Paralogs". In: *Proc. Natl. Acad. Sci. USA* 112. 10.1073/pnas.1412770112, pp. 2058–2063. DOI: 10.1073/pnas.1412770112.

Hendrick, J. P., S. L. Wolin, J. Rinke, M. R. Lerner, and J. A. Steitz (1981). "Ro small cytoplasmic ribonucleoproteins are a subclass of La ribonucleoproteins: further characterization of the Ro and La small ribonucleoproteins from uninfected mammalian cells". In: *Mol. Cell. Biol.* 1, pp. 1138–1149. DOI: 10.1128/MCB.1.12.1138.

Hernandez-Rosales, M., M. Hellmuth, N. Wieseke, K. T. Huber, V. Moulton, and P. F. Stadler (2012). "From Event-Labeled Gene Trees to Species Trees". In: *BMC Bioinformatics* 13.Suppl. 19, S6. DOI: 10.1186/1471-2105-13-s19-s6.

Hertel, J. and P. F. Stadler (2015). "The Expansion of Animal MicroRNA Families Revisited". In: *Life* 5, pp. 905–920. DOI: 10.3390/life5010905.

Heyer, R., M. Dörr, A. Jellen-Ritter, B. Späth, J. Babski, K. Jaschinski, J. Soppa, and A. Marchfelder (2012). "High throughput sequencing reveals a plethora of small RNAs including tRNA derived fragments in Haloferax volcanii". In: *RNA biology* 9.7, pp. 1011–1018. DOI: 10.4161/rna.20826.

Hiller, M., B. T. Schaar, V. B. Indjeian, D. M. Kingsley, L. R. Hagey, and G. Bejerano (2012). "A "forward genomics" approach links genotype to phenotype using independent phenotypic losses among related species". In: *Cell Rep.* 2, pp. 817–823. DOI: 10.1016/j.celrep.2012.08.032.

Hirtreiter, A., G. E. Damsma, A. C. Cheung, D. Klose, D. Grohmann, E. Vojnic, A. C. Martin, P. Cramer, and F. Werner (2010). "Spt4/5 stimulates transcription elongation through the RNA polymerase clamp coiled-coil motif". In: *Nucleic acids research* 38.12, pp. 4040–4051. DOI: 10.1093/nar/gkq135.

Höchsmann, M. (2005). "The tree alignment model: algorithms, implementations and applications for the analysis of RNA secondary structures". PhD thesis. Technische Fakultät, Universität Bielefeld.

Höchsmann, M., B. Voss, and R. Giegerich (2004). "Pure multiple RNA secondary structure alignments: a progressive profile approach". In: *IEEE/ACM Trans. Comp. Biol. Bioinf.* 1, pp. 53–62. DOI: 10.1109/TCBB.2004.11.

Hofacker, I. L., W. Fontana, P. F. Stadler, L. S. Bonhoeffer, M. Tacker, and P. Schuster (1994). "Fast folding and comparison of RNA secondary structures". In: *Monatshefte für Chemie/Chemical Monthly* 125.2, pp. 167–188. DOI: 10.1007/bf00818163.

Hoffmann, A. (2020). "The Marvellous World of tRNAs". PhD thesis. University Leipzig.

Hoffmann, S., C. Otto, S. Kurtz, C. M. Sharma, P. Khaitovich, J. Vogel, P. F. Stadler, and J. Hackermüller (2009). "Fast mapping of short sequences with mismatches, insertions and deletions using index structures". In: *PLoS computational biology* 5.9, e1000502. DOI: 10.1371/journal.pcbi.1000502.

Holland, P. W. (2013). "Evolution of homeobox genes". In: *Wiley Interdiscip Rev Dev Biol* 2, pp. 31–45. DOI: 10.1002/wdev.78.

Höner zu Siederdissen, C. (2012). "Sneaking Around concatMap: Efficient Combinators for Dynamic Programming". In: *Proceedings of the 17th ACM SIGPLAN international conference on Functional programming*. ICFP 2012. Copenhagen, Denmark: ACM, pp. 215–226. ISBN: 978-1-4503-1054-3. DOI: 10.1145/2364527.2364559. URL: http://www.bioinf.uni-leipzig.de/Software/gADP/.

Höner zu Siederdissen, C., I. L. Hofacker, and P. F. Stadler (2015a). "Product Grammars for Alignment and Folding". In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 12.3, pp. 507–519. ISSN: 1545-5963. DOI: 10.1109/TCBB.2014.2326155. URL: http://www.bioinf.uni-leipzig.de/Software/gADP/.

Höner zu Siederdissen, C., S. J. Prohaska, and P. F. Stadler (2015b). "Algebraic Dynamic Programming over General Data Structures". In: *BMC Bioinformatics* 16 Suppl 19. DOI: 10.1186/1471-2105-16-S19-S2.

Hopcroft, J. E. and J. D. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation*. 1st. Addison-Wesley Publishing Company. DOI: 10.1016/0096-0551(80)90011-9.

Huelsenbeck, J. P. and F. Ronquist (2001). "MRBAYES: Bayesian inference of phylogenetic trees". In: *Bioinformatics* 17.8, pp. 754–755. DOI: 10.1093/bioinformatics/17.8.754.

Hug, L. A., B. J. Baker, K. Anantharaman, C. T. Brown, A. J. Probst, C. J. Castelle, C. N. Butterfield, A. W. Hernsdorf, Y. Amano, K. Ise, et al. (2016). "A new view of the tree of life". In: *Nature microbiology* 1.5, p. 16048. DOI: 10.1038/nmicrobiol.2016.48.

Iacoangeli, A., T. S. Rozhdestvensky, N. Dolzhanskaya, B. Tournier, J. Schutt, J. Brosius, R. B. Denman, E. W. Khandjian, S. Kindler, and H. Tiedge (2008). "On BC1 RNA and the fragile X mental retardation protein". In: *Proc. Natl. Acad. Sci. USA* 105, pp. 734–739. DOI: 10.1073/pnas.0801034105.

Inoue, M., I. Nakamoto, K. Omae, T. Oguro, H. Ogata, T. Yoshida, and Y. Sako (2019). "Structural and Phylogenetic Diversity of Anaerobic Carbon-Monoxide Dehydrogenases". In: *Frontiers in Microbiology* 9. DOI: 10.3389/fmicb.2018.03353.

Jacox, E., C. Chauve, G. J. Szöllősi, Y. Ponty, and C. Scornavacca (2016). "ecceTERA: comprehensive gene tree-species tree reconciliation using parsimony". In: *Bioinformatics* 32.13, pp. 2056–2058. DOI: 10.1093/bioinformatics/btw105.

Jiang, L., C. Schaffitzel, R. Bingel-Erlenmeyer, N. Ban, P. Korber, R. I. Koning, D. C. de Geus, J. R. Plaisier, and J. P. Abrahams (2009). "Recycling of aborted ribosomal 50S subunit-nascent chain-tRNA complexes by the heat shock protein Hsp15". In: *Journal of molecular biology* 386.5, pp. 1357–1367. DOI: 10.1016/j.jmb.2008.10.079.

Jiang, T., L. Wang, and K. Zhang (1995). "Alignment of trees – an alternative to tree edit". In: *Theor. Comp. Sci.* 143, pp. 137–148. DOI: 10.1016/0304-3975(95)80029-9.

Jost, J. (2015). *Mathematical concepts*. Springer. DOI: 10.1007/978-3-319-20436-9.

Joyal, A. (1981). "Une théorie combinatoire des séries formelles". In: *Advances in mathematics* 42.1, pp. 1–82. DOI: 10.1016/0001-8708(81)90052-9.

Just, W. (2001). "Computational Complexity of Multiple Sequence Alignment with SP-Score". In: *J. Comp. Biol.* 8, pp. 615–623. DOI: 10.1089/106652701753307511.

Kan, T., S. Higuchi, and K. Hirata (2014). "Segmental mapping and distance for rooted labeled ordered trees". In: *Fundamenta Informaticae* 132.4, pp. 461–483. DOI: 10.1007/978-3-642-35261-4_51.

Katoh, K., K.-i. Kuma, H. Toh, and T. Miyata (2005). "MAFFT version 5: improvement in accuracy of multiple sequence alignment". In: *Nucleic Acids Res.* 33, pp. 511–518. DOI: 10.1093/nar/gki198.

Kececioglu, J. D. (1993). "The maximum weight trace problem in multiple sequence alignment". In: *Proceedings of the 4th Symposium on Combinatorial Pattern Matching*.

Vol. 684. Lecture Notes Comp. Sci. Berlin: Springer, pp. 106–119. DOI: 10.1007/bfb0029800.

Kececioglu, J. and D. Starrett (2004). "Aligning alignments exactly". In: *Proceedings of the 8th ACM Conference on Research in Computational Molecular Biology (RECOMB)*. Ed. by P. E. Bourne and D. Gusfield. New York, NY: ACM, pp. 85–96. DOI: 10.1145/974614.974626.

Kheir, E. and T. Krude (2017). "Non-coding Y RNAs associate with early replicating euchromatin in concordance with the origin recognition complex". In: *J Cell Sci.* 130, pp. 1239–1250. DOI: 10.1242/jcs.197566.

Kimura, M. et al. (1968). "Evolutionary rate at the molecular level". In: *Nature* 217.5129, pp. 624–626. DOI: 10.1038/217624a0.

Koide, T., D. J. Reiss, J. C. Bare, W. L. Pang, M. T. Facciotti, A. K. Schmid, M. Pan, B. Marzolf, P. T. Van, F.-Y. Lo, et al. (2009). "Prevalence of transcription promoters within archaeal operons and coding sequences". In: *Molecular systems biology* 5.1, p. 285. DOI: 10.1038/msb.2009.42.

Konagurthu, A. S., J. Whisstock, and P. J. Stuckey (2004). "Progressive multiple alignment using sequence triplet optimization and three-residue exchange costs". In: *J. Bioinf. and Comp. Biol.* 2, pp. 719–745. DOI: 10.1142/S0219720004000831.

Kondrak, G. (2000). "A New Algorithm for the Alignment of Phonetic Sequences". In: *Proceedings of NAACL 2000 1st Meeting of the North American Chapter of the Association for Computational Linguistics*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 288–295. DOI: 10.1.1.19.9698. URL: http://aclweb.org/anthology/A00-2038.

Koonin, E. V. (2003). "Comparative genomics, minimal gene-sets and the last universal common ancestor". In: *Nature Reviews Microbiology* 1.2, p. 127. DOI: 10.1038/nrmicro751.

Koonin, E. V. (2005). "Orthologs, paralogs, and evolutionary genomics". In: *Annu. Rev. Genet.* 39, pp. 309–338. DOI: 10.1146/annurev.genet.39.073003.114725.

Korte, B., R. Schrader, and L. Lovász (1991). *Greedoids*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-58191-5.

Kovacs, N. A., A. S. Petrov, K. A. Lanier, and L. D. Williams (2017). "Frozen in Time: The History of Proteins". In: *Molecular Biology and Evolution* 34.5, pp. 1252–1260. DOI: 10.1093/molbev/msx086.

Kristensen, D. M., Y. I. Wolf, A. R. Mushegian, and E. V. Koonin (2011). "Computational methods for Gene Orthology inference". In: *Briefings in bioinformatics* 12.5, pp. 379–391. DOI: 10.1093/bib/bbr030.

Kruskal, J. B. (1983). "An overview of sequence comparison: Time warps, string edits, and macromolecules". In: *SIAM review* 25.2, pp. 201–237. DOI: 10.1137/1025045.

Kruspe, M. and P. F. Stadler (2007). "Progressive Multiple Sequence Alignments from Triplets". In: *BMC Bioinformatics* 8, p. 254. DOI: 10.1186/1471-2105-8-254.

Kuboyama, T. (2007). "Matching and learning in trees". PhD thesis.

Kuehner, J. N., E. L. Pearson, and C. Moore (2011). "Unravelling the means to an end: RNA polymerase II transcription termination". In: *Nature reviews Molecular cell biology* 12.5, p. 283. DOI: 10.1038/nrm3098.

Kuich, W. (1997). "Semirings and formal power series: Their relevance to formal languages and automata". In: *Handbook of formal languages*. Springer, pp. 609–677. DOI: 10.1007/978-3-642-59136-5_9.

Larkin, M. A. et al. (2007). "Clustal W and Clustal X version 2.0". In: *Bioinformatics* 23, pp. 2947–2948. DOI: 10.1093/bioinformatics/btm404.

Lechner, M., S. Findeiß, L. Steiner, M. Marz, P. F. Stadler, and S. J. Prohaska (2011). "Proteinortho: Detection of (Co-)Orthologs in Large-Scale Analysis". In: *BMC Bioinformatics* 12.1, p. 124. DOI: 10.1186/1471-2105-12-124.

Lechner, M., M. Hernández Rosales, D. Doerr, N. Wieseke, A. Thévenin, J. Stoye, R. K. Hartmann, S. J. Prohaska, and P. F. Stadler (2014). "Orthology detection combining clustering and synteny for very large datasets". In: *PLoS One* 9.8, e105015. DOI: 10.1371/journal.pone.0105015.

Lee, C. (2003). "Generating consensus sequences from partial order multiple sequence alignment graphs". In: *Bioinformatics* 19, pp. 999–1008. DOI: 10.1093/bioinformatics/btg109.

Lee, C., C. Grasso, and M. F. Sharlow (2002). "Multiple sequence alignment using partial order graphs". In: *Bioinformatics* 18, pp. 452–464. DOI: 10.1093/bioinformatics/18.3.452.

Lee, D., O. Redfern, and C. Orengo (2007). "Predicting protein function from sequence and structure". In: *Nature Reviews Molecular Cell Biology* 8.12, pp. 995–1005. DOI: 10.1038/nrm2281.

Leigh, J. A., S.-V. Albers, H. Atomi, and T. Allers (2011). "Model organisms for genetics in the domain Archaea: methanogens, halophiles, Thermococcales and Sulfolobales". In: *FEMS microbiology reviews* 35.4, pp. 577–608. DOI: 10.1111/j.1574-6976.2011.00265.x.

Lemey, P., M. Salemi, and A.-M. Vandamme (2009). *The phylogenetic handbook: a practical approach to phylogenetic analysis and hypothesis testing*. Cambridge University Press. DOI: 10.1017/cbo9780511819049.

Lerner, M. R., J. A. Boyle, J. A. Hardin, and J. A. Steitz (1981). "Two novel classes of small ribonucleoproteins detected by antibodies associated with lupus erythematosus". In: *Science* 211, pp. 400–402. DOI: 10.1126/science.6164096.

Letunic, I. and P. Bork (2016). "Interactive tree of life (iTOL) v3: an online tool for the display and annotation of phylogenetic and other trees". In: *Nucleic acids research* 44.W1, W242–W245. DOI: 10.1093/nar/gkw290.

Levenshtein, V. I. (1966). "Binary codes capable of correcting deletions, insertions, and reversals". In: *Soviet physics doklady*. Vol. 10. 8, pp. 707–710.

Li, H., B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin (2009). "The sequence alignment/map format and SAMtools". In: *Bioinformatics* 25.16, pp. 2078–2079. DOI: 10.1093/bioinformatics/btp352.

Liao, D., T. Pavelitz, J. R. Kidd, K. K. Kidd, and A. M. Weiner (1997). "Concerted evolution of the tandemly repeated genes encoding human U2 snRNA (the RNU2 locus) involves rapid intrachromosomal homogenization and rare interchromosomal gene conversion". In: *EMBO J.* 16, pp. 588–598. DOI: 10.1006/geno.1995.1280.

Liao, D. (1999). "Concerted Evolution: Molecular Mechanisms and Biological Implications". In: *Am. J. Hum. Genet.* 64, pp. 24–30. DOI: 10.1086/302221.

Limbach, P. A. and M. J. Paulines (2017). "Going global: the new era of mapping modifications in RNA". In: *Wiley Interdisciplinary Reviews: RNA* 8.1, e1367. DOI: 10.1002/wrna.1367.

Lipman, D. J., S. F. Altschul, and J. D. Kececioglu (1989). "A tool for multiple sequence alignment". In: *Proc. Natl. Acad. Sci. USA* 86, pp. 4412–4415. DOI: 10.1073/pnas.86.12.4412.

Liu, S., M.-Z. Du, Q.-F. Wen, J. Kang, C. Dong, L. Xiong, J. Huang, and F.-B. Guo (2018). "Comprehensive exploration of the enzymes catalysing oxygen-involved reactions and COGs relevant to bacterial oxygen utilization". In: *Environmental Microbiology* 20.10, pp. 3836–3850. DOI: 10.1111/1462-2920.14399.

Liu, Y., J. Wang, J. Guo, and J. Chen (2012). "Complexity and parameterized algorithms for Cograph Editing". In: *Theor. Comp. Sci.* 461, pp. 45–54. DOI: 10.1016/j.tcs.2011.11.040.

Lorenz, R., S. H. Bernhart, C. Höner zu Siederdissen, H. Tafer, C. Flamm, P. F. Stadler, and I. L. Hofacker (2011a). "ViennaRNA Package 2.0". In: *Algorithms for Molecular Biology* 6.26. DOI: 10.1186/1748-7188-6-26.

Lorenz, R., S. H. Bernhart, C. H. Zu Siederdissen, H. Tafer, C. Flamm, P. F. Stadler, and I. L. Hofacker (2011b). "ViennaRNA Package 2.0". In: *Algorithms for Molecular Biology* 6.1, p. 26. DOI: 10.1186/1748-7188-6-26.

Lorenz, R., I. L. Hofacker, and S. H. Bernhart (2012). "Folding RNA/DNA hybrid duplexes". In: *Bioinformatics* 28.19, pp. 2530–2531. DOI: 10.1093/bioinformatics/bts466.

Lorenz, R., M. T. Wolfinger, A. Tanzer, and I. L. Hofacker (2016). "Predicting RNA secondary structures from sequence and probing data". In: *Methods* 103, pp. 86–98. DOI: 10.1016/j.ymeth.2016.04.004.

Lowe, T. M. and S. R. Eddy (1997). "tRNAscan-SE: A Program for Improved Detection of TransferRNA Genes in Genomic Sequence". In: *Nucleic Acids Res.* 25, pp. 955–964. DOI: 10.1093/nar/25.5.0955.

Maddison, W. P. (2000). "Testing Character Correlation using Pairwise Comparisons on a Phylogeny". In: *J. Theor. Biol.* 202, pp. 195–204. DOI: 10.1006/jtbi.1999.1050.

Maier, L.-K. and A. Marchfelder (2019). "It's all about the T: transcription termination in archaea". In: *Biochemical Society Transactions* 47.1, pp. 461–468. DOI: 10.1042/bst20180557.

Malde, K. and T. Furmanek (2013). "Increasing sequence search sensitivity with transitive alignments". In: *PloS one* 8, e54422. DOI: 10.1371/journal.pone.0054422.

Manthey, B. (2003). "Non-approximability of weighted multiple sequence alignment". In: *Theor. Comp. Sci.* 296, pp. 179–192. DOI: 10.1007/3-540-44679-6_9.

Martin, M. (2011). "Cutadapt removes adapter sequences from high-throughput sequencing reads". In: *EMBnet. journal* 17.1, pp. 10–12. DOI: 10.14806/ej.17.1.200.

Mattick, J. S. and I. V. Makunin (2006). "Non-coding RNA". In: *Human molecular genetics* 15.suppl_1, R17–R29. DOI: 10.1093/hmg/ddl046.

McBride, C. (2008). "Clowns to the left of me, jokers to the right (pearl): dissecting data structures". In: *ACM SIGPLAN Notices*. Vol. 43. 1. ACM, pp. 287–295. DOI: 10.1145/1328897.1328474.

McClintock, B. (1950). "The origin and behavior of mutable loci in maize". In: *Proceedings of the National Academy of Sciences* 36.6, pp. 344–355. DOI: 10.1073/pnas.36.6.344.

McFarlane, R. J. and S. K. Whitehall (2009). "tRNA genes in eukaryotic genome organization and reorganization". In: *Cell Cycle* 8, pp. 3102–3106. DOI: 10.4161/cc.8.19.9625.

Miklós, I. (2019). *Computational Complexity of Counting and Sampling*. Chapman and Hall/CRC. DOI: 10.1201/b22024.

Miladi, M., A. Junge, F. Costa, S. E. Seemann, J. H. Havgaard, J. Gorodkin, and R. Backofen (2017). "RNAscClust: clustering RNA sequences using structure conservation and graph based motifs". In: *Bioinformatics* 33.14, pp. 2089–2096. DOI: 10.1093/bioinformatics/btx114.

Miladi, M., E. Sokhoyan, T. Houwaart, S. Heyne, F. Costa, B. Gruening, and R. Backofen (2019). "Empowering the annotation and discovery of structured RNAs with scalable and accessible integrative clustering". In: *bioRxiv*. DOI: 10.1101/550335. eprint: https://www.biorxiv.org/content/early/2019/02/20/550335.full.pdf. URL: https://www.biorxiv.org/content/early/2019/02/20/550335.

Mirkin, B. G., T. I. Fenner, M. Y. Galperin, and E. V. Koonin (2003). "Algorithms for computing parsimonious evolutionary scenarios for genome evolution, the last universal common ancestor and dominance of horizontal gene transfer in the evolution of prokaryotes". In: *BMC evolutionary biology* 3.1, p. 2. DOI: 10.1186/1471-2148-3-2.

Möhl, M., S. Will, and R. Backofen (2010). "Lifting prediction to alignment of RNA pseudoknots". In: *J Comput Biol.* 17, pp. 429–442. DOI: 10.1089/cmb.2009.0168.

Morgan, H. L. (1970). "Spelling correction in systems programs". In: *Communications of the ACM* 13.2, pp. 90–94. DOI: 10.1145/362007.362033.

Morgenstern, B. (1999). "DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment". In: *Bioinformatics* 15, pp. 211–218. DOI: 10.1093/bioinformatics/15.3.211.

Morgenstern, B., A. Dress, and T. Werner (1996). "Multiple DNA and protein sequence alignment based on segment-to-segment comparison". In: *Proc. Natl. Acad. Sci. USA* 93, pp. 12098–12103. DOI: 10.1073/pnas.93.22.12098.

Morgenstern, B., J. Stoye, and A. W. M. Dress (1999). *Consistent Equivalence Relations: a Set-Theoretical Framework for Multiple Sequence Alignments*. Tech. rep. University of Bielefeld, FSPM. DOI: 10.1.1.37.7862.

Mosig, A., M. Guofeng, B. M. R. Stadler, and P. F. Stadler (2007). "Evolution of the Vertebrate Y RNA Cluster". In: *Th. Biosci.* 126, pp. 9–14. DOI: 10.1007/s12064-007-0003-y.

Myers, E. W., G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, M. J. Flanigan, S. A. Kravitz, C. M. Mobarry, K. H. Reinert, K. A. Remington, et al. (2000). "A whole-genome assembly of Drosophila". In: *Science* 287.5461, pp. 2196–2204. DOI: 10.1126/science.287.5461.2196.

Naidoo, K., E. Steenkamp, M. P. Coetzee, M. J. Wingfield, and B. D. Wingfield (2013). "Concerted evolution in the ribosomal RNA cistron". In: *PLoS One* 8, e59355. DOI: 10.1371/journal.pone.0059355.

Navarro, G. (2001). "A guided tour to approximate string matching". In: *ACM computing surveys (CSUR)* 33.1, pp. 31–88. DOI: 10.1145/375360.375365.

Nawrocki, E. P. and S. R. Eddy (2013). "Infernal 1.1: 100-fold faster RNA homology searches". In: *Bioinformatics* 29.22, pp. 2933–2935. DOI: 10.1093/bioinformatics/btt509.

Nawrocki, E. P., T. A. Jones, and S. R. Eddy (2018). "Group I introns are widespread in archaea". In: *Nucleic acids research* 46.15, pp. 7970–7976. DOI: 10.1093/nar/gky414.

Needleman, S. B. and C. D. Wunsch (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *J. Mol. Biol.* 48, pp. 443–453. DOI: 10.1016/0022-2836(70)90057-4.

Nei, M. (1987). *Molecular evolutionary genetics*. Columbia university press. DOI: 10.7312/nei-92038.

Nei, M. and A. P. Rooney (2005). "Concerted and Birth-and-Death Evolution of Multigene Families". In: *Annu Rev Genet* 39, pp. 121–152. DOI: 10.1146/annurev.genet.39.073003.112240.

Németh, A., J. Perez-Fernandez, P. Merkl, S. Hamperl, J. Gerber, J. Griesenbeck, and H. Tschochner (2013). "RNA polymerase I termination: Where is the end?" In: *Biochimica et Biophysica Acta (BBA)-Gene Regulatory Mechanisms* 1829.3-4, pp. 306–317. DOI: 10.1016/j.bbagrm.2012.10.007.

Newman, M. (2010). *Networks: an introduction*. Oxford university press. DOI: 10.1093/acprof:oso/9780199206650.001.0001.

Nichio, B. T., J. N. Marchaukoski, and R. T. Raittz (2017). "New tools in orthology analysis: A brief review of promising perspectives". In: *Frontiers in genetics* 8, p. 165. DOI: 10.3389/fgene.2017.00165.

Nishihara, H., A. F. Smit, and N. Okada (2006). "Functional noncoding sequences derived from SINEs in the mammalian genome". In: *Genome Res* 16, pp. 864–874. DOI: 10.1101/gr.5255506.

Notredame, C., D. G. Higgins, and J. Heringa (2000). "T-coffee: a novel method for fast and accurate multiple sequence alignment". In: *Journal of molecular biology* 302, pp. 205–217. DOI: 10.1006/jmbi.2000.4042.

Nussinov, R. and A. B. Jacobson (1980). "Fast algorithm for predicting the secondary structure of single-stranded RNA". In: *Proceedings of the National Academy of Sciences* 77.11, pp. 6309–6313. DOI: 10.1073/pnas.77.11.6309.

Nussinov, R., G. Pieczenik, J. R. Griggs, and D. J. Kleitman (1978). "Algorithms for loop matchings". In: *SIAM Journal on Applied mathematics* 35.1, pp. 68–82. DOI: 10.1137/0135006.

O'Brien, C. A., K. Margelot, and S. L. Wolin (1993). "*Xenopus* Ro ribonucleoproteins: Members of an evolutionarily conserved class of cytoplasmic ribonucleoproteins". In: *Proc. Natl. Acad. Sci. USA* 90, pp. 7250–7254. DOI: 10.1073/pnas.90.15.7250.

O'Malley, M. A. and E. V. Koonin (2011). "How stands the Tree of Life a century and a half after The Origin?" In: *Biology Direct* 6.1, p. 32. DOI: 10.1186/1745-6150-6-32.

Ochman, H., J. G. Lawrence, and E. A. Groisman (2000). "Lateral gene transfer and the nature of bacterial innovation". In: *nature* 405.6784, p. 299. DOI: 10.1038/35012500.

Ohno, S. (1970). *Evolution by gene duplication*. Springer. DOI: 10.1007/978-3-642-86659-3.

Ohta, T. (2001). "Gene families: multigene families and superfamilies". In: *e LS*. DOI: 10.1038/npg.els.0005126.

Otto, C., P. F. Stadler, and S. Hoffmann (2014). "Lacking alignments? The next-generation sequencing mapper segemehl revisited". In: *Bioinformatics* 30.13, pp. 1837–1843. DOI: 10.1093/bioinformatics/btu146.

Otto, W., P. F. Stadler, and S. J. Prohaska (2011). "Phylogenetic Footprinting and Consistent Sets of Local Aligments". In: *CPM 2011*. Ed. by R. Giancarlo and G. Manzini. Vol. 6661. Lecture Notes in Computer Science. Heidelberg, Germany: Springer-Verlag, pp. 118–131. DOI: 10.1007/978-3-642-21458-5_12.

Pawlik, M. and N. Augsten (2016). "Tree edit distance: Robust and memory-efficient". In: *Information Systems* 56, pp. 157–173. DOI: 10.1016/j.is.2015.08.004.

Perreault, J., J. P. Perreault, and G. Boire (2007). "Ro-associated Y RNAs in metazoans: evolution and diversification". In: *Mol Biol Evol* 24, pp. 1678–1689. DOI: 10.1093/molbev/msm084.

Perreault, J., J.-F. Noël, F. Brière, B. Cousineau, J.-F. Lucier, J.-P. Perreault, and G. Boire (2005). "Retropeudogenes derived from human Ro/SS-A autoantigen-associated hY RNAs". In: *Nucl. Acids Res.* 33, pp. 2032–2041. DOI: 10.1093/nar/gki504.

Peters, J. M., A. D. Vangeloff, and R. Landick (2011). "Bacterial transcription terminators: the RNA 3'-end chronicles". In: *Journal of molecular biology* 412.5, pp. 793–813. DOI: 10.1016/j.jmb.2011.03.036.

Petre, I. and A. Salomaa (2009). "Algebraic Systems and Pushdown Automata". In: *Handbook of Weighted Automata, Chapter 7*. Ed. by M. Droste, W. Kuich, and H. Vogler. Springer Berlin Heidelberg, pp. 257–289. ISBN: 978-3-642-01492-5. DOI: 10.1007/978-3-642-01492-5_7.

Pohlschröder, M. and S. Schulze (2019). "Haloferax volcanii". In: *Trends in microbiology* 27.1, pp. 86–87. DOI: 10.1016/j.tim.2018.10.004.

Porrua, O., M. Boudvillain, and D. Libri (2016). "Transcription termination: variations on common themes". In: *Trends in Genetics* 32.8, pp. 508–522. DOI: 10.1016/j.tig.2016.05.007.

Price, M. N., P. S. Dehal, and A. P. Arkin (2010). "FastTree 2–approximately maximum-likelihood trees for large alignments". In: *PloS one* 5.3, e9490. DOI: 10.1371/journal.pone.0009490.

Prohaska, S. J., S. J. Berkemer, F. Gärtner, T. Gatter, N. Retzlaff, C. H. zu Siederdissen, P. F. Stadler, et al. (2018). "Expansion of gene clusters, circular orders, and the shortest Hamiltonian path problem". In: *Journal of mathematical biology* 77.2, pp. 313–341. DOI: 10.1007/s00285-017-1197-3.

Puigbò, P., Y. I. Wolf, and E. V. Koonin (2009). "Search for a'Tree of Life'in the thicket of the phylogenetic forest". In: *Journal of biology* 8.6, p. 59. DOI: https://doi.org/10.1186/jbiol159.

Quinlan, A. R. and I. M. Hall (2010). "BEDTools: a flexible suite of utilities for comparing genomic features". In: *Bioinformatics* 26.6, pp. 841–842. DOI: 10.1093/bioinformatics/btq033.

Rautiainen, M. and T. Marschall (2017). *Aligning sequences to general graphs in $O(V+mE)$ time*. Tech. rep. bioRxiv. DOI: 10.1101/216127.

Rawlings, T. A., T. M. Collins, and R. Bieler (2003). "Changing identities: tRNA duplication and remolding within animal mitochondrial genomes". In: *Proc. Natl. Acad. Sci. USA* 100, pp. 15700–15705. DOI: 10.1073/pnas.2535036100.

Ray-Soni, A., M. J. Bellecourt, and R. Landick (2016). "Mechanisms of bacterial transcription termination: all good things must end". In: *Annual review of biochemistry* 85, pp. 319–347. DOI: 10.1146/annurev-biochem-060815-014844.

Raymann, K., C. Brochier-Armanet, and S. Gribaldo (2015). "The two-domain tree of life is linked to a new root for the Archaea". In: *Proceedings of the National Academy of Sciences* 112.21, pp. 6670–6675. DOI: 10.1073/pnas.1420858112.

Raymond, J. and P. Willett (2002). "Maximum common subgraph isomorphism algorithms for the matching of chemical structures". In: *J. Computer-Aided Mol. Design* 16, pp. 521–533. DOI: 10.1023/A:1021271615909.

Riechert, M., C. Höner zu Siederdissen, and P. F. Stadler (2016). "Algebraic Dynamic Programming for Multiple Context-Free Languages". In: *Theoretical Computer Science*. DOI: 10.1016/j.tcs.2016.05.032.

Rinaudo, P., Y. Ponty, D. Barth, and A. Denise (2012). "Tree decomposition and parameterized algorithms for RNA structure-sequence alignment including tertiary interactions and pseudoknots". In: *WABI*. Springer, pp. 149–164. DOI: 10.1007/978-3-642-33122-0_12.

Rogers, H. H., C. M. Bergman, and S. Griffiths-Jones (2010). "The evolution of tRNA genes in *Drosophila*". In: *Genome Biol Evol* 2, pp. 467–477. DOI: 10.1093/gbe/evq034.

Rogers, H. H. and S. Griffiths-Jones (2014). "tRNA anticodon shifts in eukaryotic genomes". In: *RNA* 20, pp. 269–281. DOI: 10.1261/rna.041681.113.

Rozhdestvensky, T. S., A. M. Kopylov, J. Brosius, and A. Hüttenhofer (2001). "Neuronal BC1 RNA structure: evolutionary conversion of a tRNA(Ala) domain into an extended stem-loop structure". In: *RNA* 7, pp. 722–730. DOI: 10.1017/s1355838201002485.

Rutjes, S. A., A. van der Heijden, P. H. Utz, W. J. van Venrooij, and G. J. Pruijn (1999). "Rapid nucleolytic degradation of the small cytoplasmic Y RNAs during apoptosis". In: *J. Biol. Chem.* 274, pp. 24799–24807. DOI: 10.1074/jbc.274.35.24799.

Sahyoun, A. H., M. Hölzer, F. Jühling, C. Höner zu Siederdissen, M. Al-Arab, K. Tout, M. Marz, M. Middendorf, P. F. Stadler, and M. Bernt (2015). "Towards a Comprehensive Picture of Alloacceptor tRNA Remolding in Metazoan Mitochondrial Genomes". In: *Nucleic Acids Res.* 43, pp. 8044–8056. DOI: 10.1093/nar/gkv746.

Saitou, N. and M. Nei (1987). "The neighbor-joining method: a new method for reconstructing phylogenetic trees." In: *Molecular biology and evolution* 4.4, pp. 406–425. DOI: 10.1093/oxfordjournals.molbev.a040454.

Salichos, L. and A. Rokas (2011). "Evaluating Ortholog Prediction Algorithms in a Yeast Model Clade". In: *PLoS ONE* 6, e18755. DOI: 10.1371/journal.pone.0018755.

Sanger, F., S. Nicklen, and A. R. Coulson (1977). "DNA sequencing with chain-terminating inhibitors". In: *Proceedings of the national academy of sciences* 74.12, pp. 5463–5467. DOI: 10.1073/pnas.74.12.5463.

Sankoff, D. (1972). "Matching sequences under deletion/insertion constraints". In: *Proceedings of the National Academy of Sciences* 69.1, pp. 4–6. DOI: 10.1073/pnas.69.1.4.

Sankoff, D. (1975). "Minimal mutation trees of sequences". In: *SIAM J. Appl Math.* 28, pp. 35–42. DOI: 10.1137/0128004.

Sankoff, D. and J. B. Kruskal, eds. (1983). *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison*. Reading, MA, Don Mills, Ontario: Addison-Wesley. DOI: 10.1137/1025045.

Santangelo, T. J., R. Matsumi, H. Atomi, T. Imanaka, J. N. Reeve, et al. (2008). "Polarity in archaeal operon transcription in Thermococcus kodakaraensis". In: *Journal of bacteriology* 190.6, pp. 2244–2248. DOI: 10.1128/jb.01811-07.

Santangelo, T. J. and J. N. Reeve (2006). "Archaeal RNA polymerase is sensitive to intrinsic termination directed by transcribed and remote sequences". In: *Journal of molecular biology* 355.2, pp. 196–210. DOI: 10.1016/j.jmb.2005.10.062.

Santangelo, T. J., K. M. Skinner, J. N. Reeve, et al. (2009). "Archaeal intrinsic transcription termination in vivo". In: *Journal of bacteriology* 191.22, pp. 7102–7108. DOI: 10.1128/jb.00982-09.

Santosh, B., A. Varshney, and P. K. Yadava (2014). "Non-coding RNAs: biological functions and applications". In: *Cell Biochemistry and Function* 33.1, pp. 14–22. DOI: 10.1002/cbf.3079.

Sauthoff, G., S. Janssen, and R. Giegerich (2011). "Bellman's GAP - A Declarative Language for Dynamic Programming". In: ACM, pp. 29–40. DOI: 10.1145/2003476.2003484.

Sauthoff, G., M. Möhl, S. Janssen, and R. Giegerich (2013). "Bellman's GAP – a Language and Compiler for Dynamic Programming in Sequence Analysis". In: *Bioinformatics*. DOI: 10.1093/bioinformatics/btt022.

Schirmer, S. (2011). "Comparing forests". PhD thesis. Bielefeld University.

Schirmer, S. and R. Giegerich (2011). "Forest alignment with affine gaps and anchors". In: *Combinatorial Pattern Matching*. Springer, pp. 104–117. DOI: 10.1007/978-3-642-21458-5_11.

Schirmer, S., Y. Ponty, and R. Giegerich (2014). "Introduction to RNA secondary structure comparison". In: *RNA Sequence, Structure, and Function: Computational and Bioinformatic Methods*, pp. 247–273. DOI: 10.1007/978-1-62703-709-9_12.

Schwarz, S., M. Pawlik, and N. Augsten (2017). "A New Perspective on the Tree Edit Distance". In: *International Conference on Similarity Search and Applications*. Springer, pp. 156–170. DOI: 10.1007/978-3-319-68474-1_11.

Scienski, K., J. C. F. Fay, and G. C. Conant (2015). "Patterns of Gene Conversion in Duplicated Yeast Histones Suggest Strong Selection on a Coadapted Macromolecular Complex". In: *Genome Biol Evol* 7, pp. 3249–3258. DOI: 10.1093/gbe/evv216.

Searls, D. B. (1992). "The linguistics of DNA". In: *American Scientist* 80.6, pp. 579–591.

Selkow, S. M. (1977). "The tree-to-tree editing problem". In: *Inf. Processing Let.* 6, pp. 184–186. DOI: 10.1016/0020-0190(77)90064-3.

Sellers, P. H. (1974). "An algorithm for the distance between two finite sequences". In: *Journal of Combinatorial Theory, Series A* 16.2, pp. 253–258. DOI: 10.1016/0097-3165(74)90050-8.

Shih, P. M., L. M. Ward, and W. W. Fischer (2017). "Evolution of the 3-hydroxypropionate bicycle and recent transfer of anoxygenic photosynthesis into the Chloroflexi". In: *Proceedings of the National Academy of Sciences* 114.40, pp. 10749–10754. DOI: 10.1073/pnas.1710798114.

Smith, T. F. and M. S. Waterman (1981a). "Identification of common molecular subsequences". In: *Journal of Molecular Biology* 147.1, pp. 195–197. DOI: 10.1016/0022-2836(81)90087-5.

Smith, T. F. and M. S. Waterman (1981b). "Comparison of biosequences". In: *Adv. Appl. Math.* 2, pp. 482–489. DOI: 10.1016/0196-8858(81)90046-4.

Soares, A. R. and M. Santos (2017). "Discovery and function of transfer RNA-derived fragments and their role in disease". In: *Wiley Interdiscip Rev RNA* 8, p. 5. DOI: 10.1002/wrna.1423.

Spitalny, P. and M. Thomm (2008). "A polymerase III-like reinitiation mechanism is operating in regulation of histone expression in archaea". In: *Molecular microbiology* 67.5, pp. 958–970. DOI: 10.1111/j.1365-2958.2007.06084.x.

Stamatakis, A. (2006). "RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models". In: *Bioinformatics* 22.21, pp. 2688–2690. DOI: 10.1093/bioinformatics/btl446.

Stamatakis, A. (2014). "RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies". In: *Bioinformatics* 30.9, pp. 1312–1313. DOI: 10.1093/bioinformatics/btu033.

Steiner, L., P. F. Stadler, and M. Cysouw (2011). "A Pipeline for Computational Historical Linguistics". In: *Language Dynamics & Change* 1, pp. 89–127. DOI: 10.1163/221058211X570358.

Stoye, J., V. Moulton, and A. W. M. Dress (1997). "DCA: an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment". In: *Comput. Appl. Biosci.* 13, pp. 625–626. DOI: 10.1093/bioinformatics/13.6.625.

Straub, J., M. Brenneis, A. Jellen-Ritter, R. Heyer, J. Soppa, and A. Marchfelder (2009). "Small RNAs in haloarchaea: identification, differential expression and biological function". In: *RNA biology* 6.3, pp. 281–292. DOI: 10.4161/rna.6.3.8357.

Sun, F. J., S. Fleurdépine, C. Bousquet-Antonelli, G. Caetano-Anollès, and J. M. Deragon (2007). "Common evolutionary trends for SINE RNA structures". In: *Trends Genet.* 23, pp. 26–33. DOI: 10.1016/j.tig.2006.11.005.

Swofford, D. L. (2004). "Paup*: Phylogenetic analysis using parsimony (and other methods)". In: *Dictionary of Bioinformatics and Computational Biology*. DOI: 10.1002/0471650129.dob0522.

Tafer, H. and I. L. Hofacker (2008). "RNAplex: a fast tool for RNA–RNA interaction search". In: *Bioinformatics* 24.22, pp. 2657–2663. DOI: 10.1093/bioinformatics/btn193.

Tai, K.-C. (1979). "The tree-to-tree correction problem". In: *Journal of the ACM (JACM)* 26.3, pp. 422–433. DOI: 10.1145/322139.322143.

Tatusov, R. L., E. V. Koonin, and D. J. Lipman (1997). "A genomic perspective on protein families". In: *Science* 278, pp. 631–637. DOI: 10.1126/science.278.5338.631.

Tendeau, F. (1998). "Computing abstract decorations of parse forests using dynamic programming and algebraic power series". In: *Theoretical computer science* 199.1-2, pp. 145–166. DOI: 10.1016/s0304-3975(97)00271-5.

Teshima, K. M. and H. Innan (2004). "The Effect of Gene Conversion on the Divergence Between Duplicated Genes". In: *Genetics* 166, pp. 1553–1560. DOI: 10.1534/genetics.166.3.1553.

Teunissen, S. W. M., M. J. M. Kruithof, A. D. Farris, J. B. Harley, W. J. van Venrooij, and G. J. M. Pruijn (2000). "Conserved features of Y RNAs: a comparison of experimentally derived secondary structures". In: *Nucl. Acids Res.* 28, pp. 610–619. DOI: 10.1093/nar/28.2.610.

Thiel, B. C., C. Flamm, and I. L. Hofacker (2017). "RNA structure prediction: from 2D to 3D". In: *Emerging Topics in Life Sciences* 1.3, pp. 275–285. DOI: 10.1042/etls20160027.

Thomason, M. K. and G. Storz (2010). "Bacterial antisense RNAs: how many are there, and what are they doing?" In: *Annual review of genetics* 44, pp. 167–188. DOI: 10.1146/annurev-genet-102209-163523.

Thomm, M., W. Hausner, and C. Hethke (1993). "Transcription factors and termination of transcription in Methanococcus". In: *Systematic and applied microbiology* 16.4, pp. 648–655. DOI: 10.1016/s0723-2020(11)80336-x.

Tiepmar, J. and G. Heyer (2017). "An Overview of Canonical Text Services". In: *Linguistics Literature Studies* 5, pp. 132–148. DOI: 10.13189/lls.2017.050209.

Vaddadi, K., N. Sivadasan, K. Tayal, and R. Srinivasan (2017). *Sequence Alignment on Directed Graphs*. Tech. rep. bioRxiv. DOI: 10.1101/124941.

Velandia-Huerto, C. A., S. J. Berkemer, A. Hoffmann, N. Retzlaff, L. C. Romero Marroquín, M. Hernández Rosales, P. F. Stadler, and C. I. Bermúdez-Santana (2016). "Orthologs, turn-over, and remolding of tRNAs in primates and fruit flies". In: *BMC Genomics* 17, p. 617. DOI: 10.1186/s12864-016-2927-4.

Viterbi, A. (1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *IEEE Transactions on Information Theory* 13.2, pp. 260–269. DOI: 10.1109/TIT.1967.1054010.

Wagner, R. A. and M. J. Fischer (1974). "The string-to-string correction problem". In: *Journal of the ACM (JACM)* 21.1, pp. 168–173. DOI: 10.1145/321796.321811.

Walker, J. E., O. Luyties, and T. J. Santangelo (2017). "Factor-dependent archaeal transcription termination". In: *Proceedings of the National Academy of Sciences* 114.33, E6767–E6773. DOI: 10.1073/pnas.1704028114.

Wang, L. and T. Jiang (1994). "On the complexity of multiple sequence alignment". In: *J Comput Biol* 1, pp. 337–348. DOI: 10.1089/cmb.1994.1.337.

Wang, P. P. and I. Ruvinsky (2012). "Family size and turnover rates among several classes of small non-protein-coding RNA genes in *Caenorhabditis* nematodes". In: *Genome Biol Evol* 4, pp. 565–574. DOI: 10.1093/gbe/evs034.

Wang, Z., M. Gerstein, and M. Snyder (2009). "RNA-Seq: a revolutionary tool for transcriptomics". In: *Nature reviews genetics* 10.1, p. 57. DOI: 10.1038/nrg2484.

Ward, N. and G. Moreno-Hagelsieb (2014). "Quickly Finding Orthologs as Reciprocal Best Hits with BLAT, LAST, and UBLAST: How Much Do We Miss?" In: *PLoS ONE* 9, e101850. DOI: 10.1371/journal.pone.0101850.

Wareham, H. T. (1995). "A simplified proof of the NP- and MAX SNP-hardness of multiple sequence tree alignment". In: *J Comput Biol.* 2, pp. 509–514. DOI: 10.1089/cmb.1995.2.509.

Watanabe, Y.-i. and S. Yoshinari (2013). "Intron and RNA splicing in Archaea". In: *Viva Orig* 41.1213, p. 18.

Weiss, M. C., F. L. Sousa, N. Mrnjavac, S. Neukirchen, M. Roettger, S. Nelson-Sathi, and W. F. Martin (2016). "The physiology and habitat of the last universal common ancestor". In: *Nature Microbiology* 1.9, p. 16116. DOI: 10.1038/nmicrobiol.2016.116.

Woese, C. (1998). "The universal ancestor". In: *Proceedings of the National Academy of Sciences* 95.12, pp. 6854–6859. DOI: 10.1073/pnas.95.12.6854.

Woese, C. R. (1987). "Bacterial evolution." In: *Microbiological reviews* 51.2, p. 221.

Woese, C. R. and G. E. Fox (1977). "The concept of cellular evolution". In: *Journal of Molecular Evolution* 10.1, pp. 1–6. DOI: 10.1007/bf01796132.

Woese, C. R., O. Kandler, and M. L. Wheelis (1990). "Towards a natural system of organisms: proposal for the domains Archaea, Bacteria, and Eucarya." In: *Proceedings of the National Academy of Sciences* 87.12, pp. 4576–4579. DOI: 10.1073/pnas.87.12.4576.

Wolfe, J. M. and G. P. Fournier (2018). "Horizontal gene transfer constrains the timing of methanogen evolution". In: *Nature Ecology & Evolution* 2.5, pp. 897–903. DOI: 10.1038/s41559-018-0513-7.

Wolff, J. G. (2000). "Syntax, parsing and production of natural language in a framework of information compression by multiple alignment, unification and search". In: *J. Universal Comp. Sci.* 6.8, pp. 781–829. DOI: 10.3217/jucs-006-08-0781.

Yorgey, B. A. (2010). "Species and functors and types, oh my!" In: *ACM Sigplan Notices*. Vol. 45. 11. ACM, pp. 147–158. DOI: 10.1145/1863523.1863542.

Yorgey, B. A., S. Weirich, and J. Carette (2014). "Labelled structures and combinatorial species". In:

Yorgey, B. A. (2014). *Combinatorial species and labelled structures*. University of Pennsylvania.

Zaremba-Niedzwiedzka, K. et al. (2017). "Asgard archaea illuminate the origin of eukaryotic cellular complexity". In: *Nature* 541.7637, pp. 353–358. DOI: 10.1038/nature21031.

Zhang, K. and D. Shasha (1989). "Simple fast algorithms for the editing distance between trees and related problems". In: *SIAM J Computing* 18, pp. 1245–1262. DOI: 10.1137/0218082.

Zuckerkandl, E. and L. Pauling (1965). "Molecules as documents of evolutionary history". In: *Journal of theoretical biology* 8.2, pp. 357–366. DOI: 10.1016/0022-5193(65)90083-4.

Zuker, M. and P. Stiegler (1981). "Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information". In: *Nucleic acids research* 9.1, pp. 133–148. DOI: 10.1093/nar/9.1.133.

Zvelebil, M. J. and J. O. Baum (2007). *Understanding bioinformatics*. Garland Science. DOI: 10.1201/9780203852507.

Curriculum Scientiae

Personal Information

Name	Sarah J. Berkemer
Birthday	March 23, 1990
Birthplace	Speyer, Germany
Institutions	• Bioinformatics Group, Department of Computer Science, Leipzig University, Härtelstr. 16-18, 04107 Leipzig, Germany • Max Planck Institute for Mathematics in the Sciences (MPI MIS), Inselstraße. 22, 04103 Leipzig, Germany
E-mail	bsarah@bioinf.uni-leipzig.de
Webpage	www.bioinf.uni-leipzig.de/∼bsarah

Education

04/2015 - 06/2019	**PhD Student** • International Max Planck Research School (IMPRS) at MPI MIS Leipzig and • Bioinformatics Group, Department of Computer Science, Leipzig University
06/2018 - 07/2018	Complex Systems Summer School 2018 • Santa Fe Institute for Complex Systems (SFI), Santa Fe, NM, USA
06/2017 - 09/2017	JSPS Summer Program 2017 • Earth-Life-Science Institute (ELSI), Tokyo Institute of Technology, Tokyo, Japan

Education (continued)

04/2013 - 03/2015	**Master of Science**, Bioinformatics • Universität Leipzig, Leipzig, Germany • Thesis: Processed Small RNAs in Archaea and BHB Elements.
05/2014 - 08/2014	Google Summer of Code 2014 • Project: Transalign: Open Source high-performance Biohaskell. Convert a program used to find transitive alignments to high-performance Haskell code.
08/2012 - 01/2013	Semester abroad (Erasmus Program) • Linköping University, Linköping, Sweden
10/2009 - 08/2012	**Bachelor of Science**, Bioinformatics • Saarland University, Saarbrücken, Germany • Thesis: Cograph Editing: An Approach to Adjust the Orthology Relation for the Reconstruction of Phylogenetic Trees
2009	German-French highschool diploma (Abi-Bac) • Ludwigshafen am Rhein, Germany

Teaching

09/2016	Teaching *Introductory bioinformatics course for biologists*, Jakarta, Indonesia
03/2015	Teaching Assistant Spring school *Programming for evolutionary biology*, Leipzig, Germany
since 10/2015	Teaching Assistant Bachelor and Master courses for Computer Science and Bioinformatics, Universität Leipzig

Languages

German:	native speaker
English:	fluent
French:	good working knowledge

Selbständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angeführten Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, und alle Angaben, die auf mündlichen Auskünften beruhen, als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialen oder erbrachten Dienstleistungen als solche gekennzeichnet.

(Ort, Datum)

(Unterschrift)