Karlsruhe Institute of Technology

Technical and Algorithmic Optimization of PaPaRa

Master Thesis of

Johanna Wegmann

At the Department of Informatics
Institute of Theoretical Informatics

Reviewer:	Prof. Dr. Alexandros Stamatakis
	Prof. Dr.-Ing. Tamim Asfour
Advisor:	M.Sc. Pierre Barbera
	M.Sc. Sarah Lutteropp

Time Period: 01.06.2019 – 11.11.2019

Statement of Authorship

I hereby declare that this document has been composed by me and describes my own work, unless otherwise acknowledged in the text.

Place, Date

. . . .

(**Name**)

Abstract

Phylogenetics, the study of evolutionary relationships between organisms, uses, among other things, algorithms for the placement of genetic data into existing phylogenetic trees. One tool, which enables phylogenetic placement by providing alignments of given DNA sequences to references in an existing phylogeny, is *PaPaRa*. Like other alignment software, *PaPaRa* performs a large amount of calculations. This thesis proposes and analyzes methods for reducing the number of alignment operations, thus accelerating *PaPaRa* with limited sacrifices in the quality of the result. The discussed screening methods rely on sequence abstractions that count DNA fragments of specific length k called k-mers.

Deutsche Zusammenfassung

Phylogenetik, die Wissenschaft der evolutionären Verwandschaftsbeziehungen zwischen Organismen, benötigt Methoden um genetisches Material in phylogenische Bäume einzuordnen. *PaPaRa* ist ein Werkzeug, das die Einordnung ermöglicht indem es für gegebene DNA-Sequenzen Alignments zu Referenzsequenzen in Vererbungsbäumen erzeugt. Wie andere Alignment Software auch ist *PaPaRa* sehr rechenaufwändig. Diese Arbeit betrachtet und analysiert diverse k-mer basierte Verfahren, welche die Anzahl an Alignmentprozesse reduzieren und damit beschleunigen, ohne große Qualitätsabstriche. Die in dieser Arbeit diskutierten Methoden basieren auf Sequenzabstraktionen, die DNA Fragmente der Länge k zählen und k-mer genannt werden.

List of Figures

List of Tables

Contents

Chapter 1 Introduction

The field of bioinformatics applies tools and methods of computer science to further the understanding of biology. One main focus is analyzing genetic code that lies inside the *deoxyribonucleic acid* (DNA) present in organism cells. DNA encodes all structural and functional information needed for the existence of organisms. Chemically, it is a macro-molecule consisting of two intertwining strands that form the well-known helix structure. Each strand is created by concatenating four smaller building blocks, namely, the bases *Adenine* (A), *Thymine* (T), *Cytosine* (C) and *Guanine* (G). Each of these *nucleotides* is connected to a partner on the opposite strand and forms fixed base pairs of A-T or C-G respectively.

The order of the base pairs is the decisive information and can be extracted from the physical molecule by *DNA sequencing.* These laboratory methods produce so called *reads* or *sequence strings*, which are fragments of the investigated DNA. The most popular sequencing approaches [SNC77; Kne+19] artificially trigger the biological process of DNA replication outside the cell. Nucleotides added during the process are modified such that they can be optically identified in the resulting duplicate.
Around the year 2008, enhancements and changes to the original sequencing process emerged and are summarized with the term *next generation sequencing* (NGS) [SJ08]. Technological advances led to significantly higher throughput rates and lower costs per read, caused mainly by new methods for simultaneous execution of multiple sequencing processes. This is best demonstrated by the cost for sequencing the 3 billion base pairs of the human genome dropping from \$10 million in 2001 to currently \$1000 [Wet]. Another application made possible by NGS is the sequencing of *meta-genomes*, which refers to all genetic material present in an environmental sample. In contrast to traditional methods NGS does not rely on separation and cultivation of individual samples and therefore a greater diversity of species can be analyzed.

Now the extracted sequences can be analyzed, for example, by assessing the similarity to other species with the help of phylogenies. A *phylogeny* is a tree structure and describes the evolutionary relationships between species. Every leaf node (*taxon*) represents one species, inner nodes can be interpreted as a common *ancestral* species. The branch lengths are an indicator for the time span it took for an ancestor to mutate to a child species. Since the true evolution is unknown all phylogenies are hypothetical. One use of phylogenies is the identification of related virus variants, for example, to predict the course and possible treatments of the Ebola virus epidemic in Sierra Leone [Par+15].

In contrast to previous methods, where the phylogeny was created from scratch, now the focus of *phylogenetic placement* lies on inserting large amounts of anonymous meta-genome sequences into a known *reference tree*. The *Evolutionary Placement Algorithm* (EPA) [Bar+18] for example temporarily places the sequence in question into one branch at a time, then compares and evaluates the resulting trees. The best insertion point found is then used to classify the read.
This yields a qualitative improvement over methods simply comparing nucleotide sequence strings [KG01] in cases where no sufficiently close relatives are documented in a genetic database. The obtained placements are especially useful as a tool for analyzing the structure of meta-genomes. For example, [ME13] created a method to compare distributions of small meta-genomes among species and could confirm a connection between an imbalance of otherwise harmless bacteria types and a bacterial vaginosis infection [Sri+12].
In the course of phylogenetic placement, first, the region of the DNA, from which the sequence in question may originate from, is identified. This is done by finding the best position for the identified nucleotides in the investigated DNA, such that it has the highest overlap with it. This process is called *alignment*. The part of the DNA not matched by the smaller sequence is marked by so called *gap characters*, such that the sequence with the gap characters and the DNA have the same length in the end (e.g., see Figure 2.1).

This thesis focuses on the *PaPaRa* program which aims to align thousands of short sequences to already aligned reference sequences. *PaPaRa*'s central idea is to incorporate the phylogeny of the references into the alignment process.
Chapter 2 provides an introduction into the bioinformatics concepts needed in the following chapters. Then, the *PaPaRa* algorithm is defined in Chapter 3. The new idea for reducing the computation time of *PaPaRa* with the help of *k-mer heuristics* is explained in Chapter 5. The respective implementation is discussed in Chapter 6. The methods for the evaluation of the new approach are described in Chapter 7. The respective results are presented in Chapter 8. Chapter 9 discusses future work and provides a conclusion.

Chapter 2 General Bioinformatics Concepts

Section 2.1 Related Work

PaPaRa is used to fit DNA sequences to reference sequences, called *alignment*. There exists a plethora of alternative alignment tools as shown in [Wik19b]. A short introduction to different concepts is given here.
One of the oldest Bioinformatics algorithms is the *basic local alignment search tool* (BLAST, [Alt+90]), which is still used to search for sequences in genetic databases. It also uses a heuristic to find small matches, then looks in their surroundings for further resemblances. However, as [KG01] suggested, BLAST does not always find the closest relative - opposed to the better performance of *PaPaRa*.
Through technical advances, the amount of sequences that need identification rises and so does the need for performant solutions. Parallel implementations are realized in *PaPaRa* and others [Dai16; Rog11] by using the *single instruction multiple data* (*SIMD*) principle, realized through designated CPU instructions. The authors of [Ruc+18] used programmable hardware for speed up. There are also methods that create indexes on the reference sequences to find relevant matching positions faster [LS12].

In this thesis, an alignment-free heuristic is created to screen alignment possibilities. Therefore, the sequences are translated to vectors, counting the frequency of fragments of fixed length. Most alignment-free methods rely on this strategy. An alternative is also introduced here: *Chaos Game Representation* [Jef90] is a graphical approach of representing sequences, by iteratively adding the nucleotides. It constructs an image by traversing a 2-dimensional plane, whereby each added nucleotide dictates the direction of the subsequent point on the image. The constructed images can then be interpreted and used to compare the corresponding sequences [Alm+01].

Section 2.2 Bioinformatics Terminology

A short overview over common Bioinformatics expressions is given in the following:

sequence	A chain of DNA characters written as a string.
sequencing	The DNA extraction process, which results in a sequence.
site	A column of a alignment.
phylogeny	A tree that conveys evolutionary relationships between organisms.
taxon	A group of organisms, e.g. a species. Also, leafs of the phylogeny.

Section 2.3 Dynamic Programming Alignment

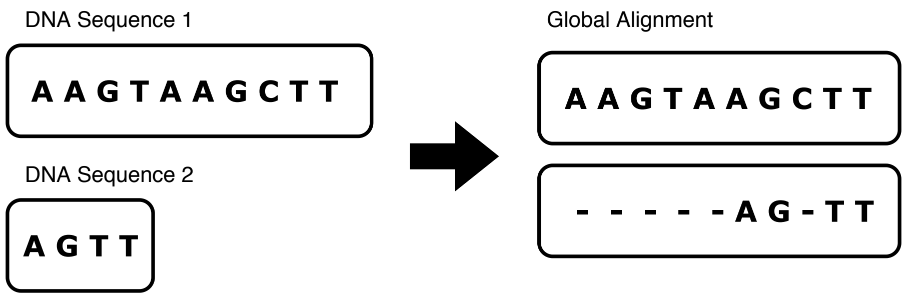

Figure 2.1: Global alignment process. Global alignment describes the process of finding the best fit of the short DNA sequence “AGTT” to the longer DNA sequence “AAGTAAGCTT” by inserting gaps.

A well known problem in Bioinformatics to find a *global alignment* of two sequences, which strives to identify similar regions and aligns them by inserting gaps into both sequences (see Figure 2.1). In general this is solved with the help of a methodology called *dynamic programming*, which divides a problem into overlapping sub problems and reduces the computational effort by caching partial solutions.

Figure 2.2 sketches the idea of the *Needleman-Wunsch algorithm* [NW70] - one of the first dynamic programming alignment algorithms which also forms the basis for the alignment process used in *PaPaRa*.

During the setup, the two sequences are written alongside a table. The longer one ($L := AACT$) is usually on top, filling the columns, and the characters of the other ($V := AGT$) usually at the side, filling the rows.
Then a column and row of initial values are added in front. In the example, they start with 0 and decrease by one per slot.
Now, the table is filled cell by cell starting from the top left corner and ending in the bottom right corner, creating the similarity matrix H. Each entry describes the best score that can be reached by aligning the prefixes of both queries to each other. So cell $H^{i,j}$ contains the best score for aligning L from index 0 to j with V from index 0 to i. The crucial ingredient is the recursion step, a rule for calculating a new cell $H^{i,j}$ value from the surrounding ones is as follows:

$$
\begin{aligned}
H^{i,j} &= \max \begin{cases} H^{i,j}_{pot1} = H^{i-1,j} & + \; GP \\ H^{i,j}_{pot2} = H^{i,j-1} & + \; GP \\ H^{i,j}_{pot3} = H^{i-1,j-1} & + \; S^{i,j} \end{cases} \\
GP &= -1 \\
S^{i,j} &= \begin{cases} 1 & \text{if } V^i \text{ and } L^j \text{ match} \\ -1 & \text{otherwise} \end{cases}
\end{aligned}
\tag{2.1}
$$

The gap penalty GP is the punishment for inserting a gap. $S^{i,j}$ rewards the alignment of nucleotides of the same type with 1 and punishes a mismatch with -1. Therefore, the nucleotides at positions i in V and j in L have to be compared. For the score of $H^{i,j}$, three different ways to align the sub sequences have to be considered. The arrows $\uparrow$, $\nwarrow$ and $\leftarrow$ show from which H entry, $H^{i,j}$ is recursively calculated. Afterwards the three possible alignments are scored using $H^{i,j}_{pot}$. $L^{2,2}_{pot}$ and $V^{2,2}_{pot}$ are potential candidates for aligning the nucleotides of V from $i : 0-2$ and of L from $j : 0-2$, in doing so gaps can be inserted.

In the example Figure 2.2, the score $H^{2,2}$ for the alignment of AG with AA is the maximum out of the following 3 alternatives:

$\uparrow$ corresponds to the first line in the maximization of Equation 2.1. It takes the score above $H^{1,2}$, adds a gap penalty GP and results in $H^{2,2}_{pot1} = 0 - 1 = -1$. This is equivalent to inserting a gap into L after the character of the current column.

$$
\begin{aligned}
L^{2,2}_{pot1} &= L_{\uparrow} + \_ = \text{AA\_} \\
V^{2,2}_{pot1} &= V_{\uparrow} + \text{G} = \text{A\_G}
\end{aligned}
$$

$\nwarrow$ adds a match reward or mismatch penalty $S^{2,2}$ to the diagonal value $H^{1,1}$ and gives $H^{2,2}_{pot2} = 1 - 1 = 0$. For the alignment construction, add the respective character to the previous sequence.

$$
\begin{aligned}
L^{2,2}_{pot2} &= L_{\nwarrow} + \text{A} = \text{AA} \\
V^{2,2}_{pot2} &= V_{\nwarrow} + \text{G} = \text{AG}
\end{aligned}
$$

$\leftarrow$ Similar to the first alternative ($\uparrow$), with the difference that the gap is inserted into V instead of L. ($H^{2,2}_{pot3} = 0 - 1 = -1$)

$$
\begin{aligned}
L^{2,2}_{pot3} &= L_{\leftarrow} + \text{A} = \text{A\_A} \\
V^{2,2}_{pot3} &= V_{\leftarrow} + \_ = \text{AG\_}
\end{aligned}
$$

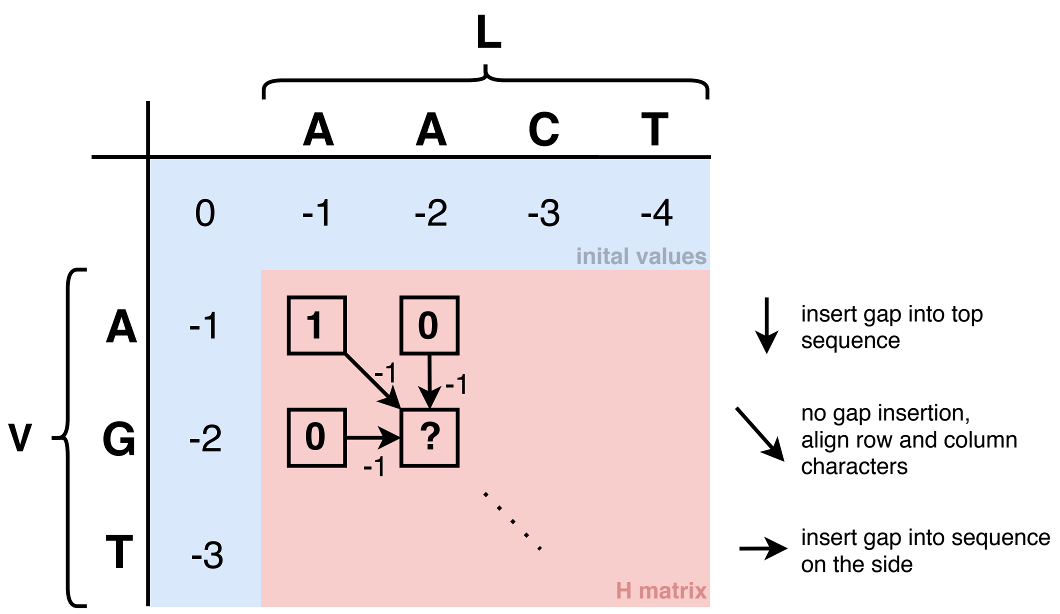
L
A A C T
0 -1 -2 -3 -4
inital values
V
A -1
G -2
T -3
1
0
0
?
-1
-1
-1
H matrix
insert gap into top sequence
no gap insertion, align row and column characters
insert gap into sequence on the side

Figure 2.2: Needleman-Wunsch Recursion Step. Alignment of “AG” to “AA” from “AGT” and “AACT”.

The maximum value of the three alternatives above is selected and stored in the matrix cell $H^{2,2} = 0$ and additionally the related arrow is reversed $\nwarrow$ and then stored for backtracking.

One can see the dynamic programming idea now: the cell values are reused multiple times and are the cached partial solutions of a recursive alignment algorithm. The best alignment is now constructed by backtracking the stored arrows from the bottom right to the top left and using the above outlined associated sequence build up for each step.

Chapter 3 PaPaRa

Pa*rsimony-based* ***P****hylogeny-****a****ware short* ***R****ead* **a***lignment*, henceforth called *PaPaRa*, is a tool, first released in 2011 by Simon Berger [BS11], which is used for the alignment of short DNA sequences to already existing reference alignments.

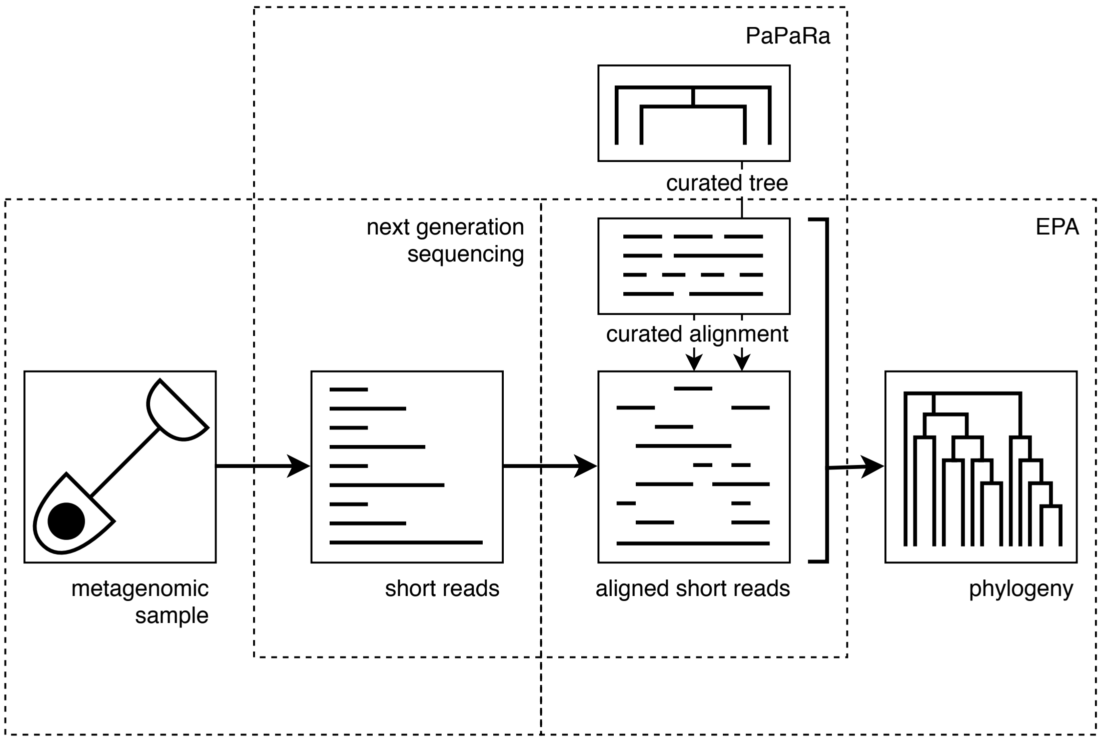

Figure 3.1: ***PaPaRa*** **in the context of other tools.** A environmental, metagenomic sample is taken. Short DNA sequences are read with the help of next generation sequencing. *PaPaRa* takes those reads, a reliable reference alignment and the associated tree and produces an alignment for the short reads. The aligned reads are then used as input for phylogenetic placement tools, such as EPA.

Due to the emergence of next generation sequencing techniques since the beginning of the century, large amounts of rather short sequences between 100 and 600 *base pairs* (bp) [Jün+13] are extracted. Still, the DNA regions that these sequences belong to need to be identified. In addition to the difficulties introduced by large quantities and high fragmentation the samples are *metagenomic*, meaning that they stem from various potentially unknown species.
Figure 3.1 shows *PaPaRa*'s importance as an intermediate step, preparing those short reads for the classification into evolutionary relationships, called phylogenies. It requires a confident and well-curated reference tree (RT) and an associated *reference alignment* (RA) as a model of reality. It extracts the underlying and implicit correlations between them and emulates these in the resulting alignment. The hereby created combined alignment of short sequences, called *queries* (QS), and the RA are fed into a separate tool for phylogenetic placement, for example EPA-ng [Bar+18] (see Chapter 1).

PaPaRa's process consists of four steps that are explained in the following sections: First, *PaPaRa* creates hypothetical *ancestral sequences* for each tree edge. With the help of a gap evolution model, it simultaneously assesses how probable gaps in each ancestral site are. Then, a dynamic programming algorithm aligns each query pairwise against every ancestor with additional gap information and remembers the highest similarity score. Lastly, backtracking the best pairs creates the aligned query sequences.

Section 3.1 Ancestral Sequence Creation

PaPaRa encodes topological information in ancestral state sequences and aligns query sequences against those in later steps. The following describes how they are created.

One starts with an *unrooted* reference tree with all r reference alignment sequences as the leaves. For every branch of the overall $2r - 3$ branches *PaPaRa* computes an associated ancestral sequences. When a virtual root is inserted into a specific branch a *rooted* tree is created. The rooted tree on the right in Figure 3.2 is the result of adding a root to the branch leading to leaf 2 of the unrooted tree on the left.

The *ancestral sequence* is located at that root and one can think of it as a representation of all possible, most probable, common ancestors for all taxa given the evolutionary relationships as defined by that specific tree.
For its creation, one starts with 2 leaf nodes and creates a combined sequence, a *profile*, at the node connecting both. This is repeated for every inner node and emerges into the ancestral sequence at the root. This process can create sequences with sites that represent multiple nucleotides, which make them ambiguous.

As an example, the ancestor of the branch leading to leaf 2 of the original tree shown on the left corner of Figure 3.2 is prepared:
First create the inner sequence (I) by combining (3) and (4), next (II) is formed from (1) and (I) and the final result is (III). In the combining step, the parent profile is formed for each site separately by choosing either the intersection of the child nucleotides if it is not empty or the union otherwise. In the example sequence (II) "A AC A" (meaning: A at site 1, A or C at site 2, and A at site 3) and sequence (2) "A C T" are combined and result in sequence "A C A", because $\{A\} = \{A\} \cap \{A\}$, $\{C\} = \{A, C\} \cap \{C\}$, $\{A, T\} = \{A\} \cup \{T\}$.

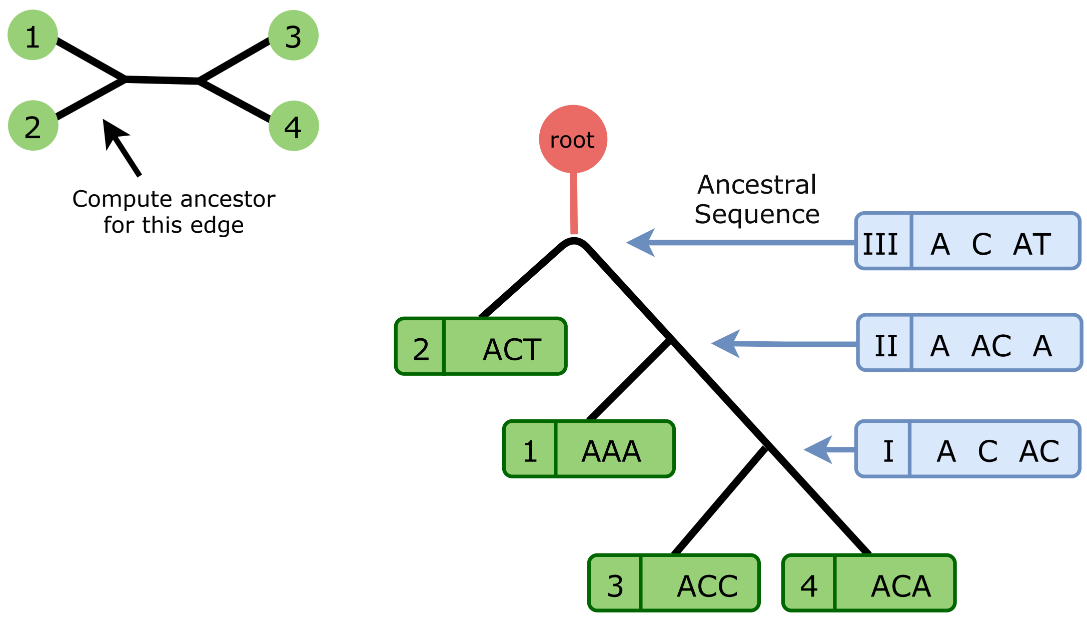

Figure 3.2: Exemplified creation of an ancestral sequence. The ancestor for the branch leading to leaf 2 is created by combining sequences connected to the same parent bottom up.

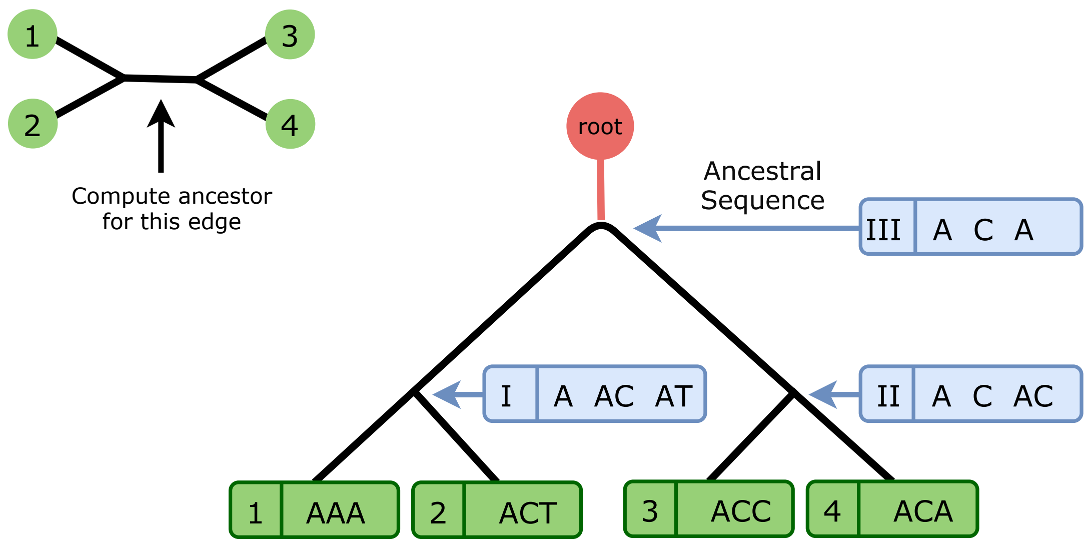

Figure 3.3: Exemplified creation of the second ancestral sequence.

The created profile sequence represents all combinations that can be built by concatenating the sets, here: “ACA” and “ACT”. If only unions were applied for each site, the ancestral state sequences would be the same for every edge, but intersecting them ensures that the minimal required number of mutations is accounted for to explain the data. This is modelled from the *parsimony criterion* described in [Fit71]. Figure 3.3 shows, how a root inserted into a another branch leads to a different ancestor.

Section 3.2 Gap Signal Propagation

During ancestral sequence creation, a corresponding *gap flag vector* is computed which specifies a probability for each site to contain a gap.
Analogous to the ancestral state vector, the gap vector for each edge is built recursively, from the leaves to the root, by combining the gap vectors of both children. A node x holds two likelihood vectors: $L_{gap}^{(x)}$ denoting a sequence of site-wise probability of gap occurrence and $L_{no\text{-}gap}^{(x)}$ the same for no gaps. For the leafs L_{gap} is initialized as a sequence of 1s and 0s, with a 1 if the leaf sequence site contains a gap and 0, otherwise. The no-gap probability $L_{no\text{-}gap}^{(x)}$ is inverse to $L_{gap}^{(x)}$. To track how the manifestation of gaps changes along a branch, a *continuous-time Markov process* is used. It is described by the states *gap* and *no-gap* and derives the state transition probability matrix $P(t)$. For details see [Ste09].

$$P(t) = e^{Qt} = \begin{vmatrix} P_{gap\to gap}(t) & P_{gap\to no\text{-}gap}(t) \\ P_{no\text{-}gap\to gap}(t) & P_{no\text{-}gap\to no\text{-}gap}(t) \end{vmatrix} \tag{3.1}$$

It uses the instantaneous transition rate matrix Q and the prior gap probabilities π_{gap} and $\pi_{no\text{-}gap}$, where π_{gap} is the percentage of gaps in the original reference alignment and $\pi_{no\text{-}gap} = 1 - \pi_{gap}$:

$$Q = \begin{vmatrix} -\pi_{gap} & \pi_{gap} \\ \pi_{no\text{-}gap} & -\pi_{no\text{-}gap} \end{vmatrix} \tag{3.2}$$

Now, the probability of a site changing from gap to no-gap in time t, $gap \to no\text{-}gap(t)$, can be calculated by computing a concrete, time-specific transition matrix through inserting t into e^{Qt} and selecting the specific index. The combining step, based on the *Felsenstein pruning algorithm* from [Fel81], uses the time reversibility property of Markov processes and computes the likelihoods of a parent p containing gaps in a rooted tree from the likelihoods of both children c_1 and c_2:

$$L_{k\in\{gap,no\text{-}gap\}}^{(p)} = \Big(\sum_{i\in\{gap,no\text{-}gap\}} P_{k\to i}(t_1)L_i^{(c_1)}\Big)\Big(\sum_{j\in\{gap,no\text{-}gap\}} P_{k\to j}(t_2)L_j^{(c_2)}\Big) \tag{3.3}$$

The evolution time t_1 from the gap sequence at p to the gap sequence at c_1 is the branch length between the nodes as taken from the phylogeny (t_2 analogously). After applying this step recursively, the likelihood vector at the root is translated into gap flag vectors to limit the computational effort. The majority rule derives 1 if $L_{gap} > L_{no\text{-}gap}$ and 0 otherwise. When implementing Equation 3.1, *PaPaRa* performs Eigenvalue decomposition on matrix Q, which results in a matrix of columns of eigenvectors U and a diagonal matrix of eigenvalues Λ.

$$Q = U\Lambda U^{-1} \tag{3.4}$$

Since e to the power of a diagonal matrix requires raising each diagonal value separately, $P(t)$ is now easy to compute.

$$P(t) = e^{Qt} = e^{U\Lambda U^{-1}t} = Ue^{\Lambda t}U^{-1} \tag{3.5}$$

$$e^{\Lambda t} = \exp\begin{vmatrix} \lambda_1 t & 0 \\ 0 & -\lambda_2 t \end{vmatrix} = \begin{vmatrix} e^{\lambda_1 t} & 0 \\ 0 & e^{\lambda_2 t} \end{vmatrix} \tag{3.6}$$

For more details on matrix exponentials see [HJ12].

Section 3.3 Alignment

Now, a query sequence can be aligned to an ancestral sequence as calculated in Section 3.1 with additional information from the gap flag vector. To achieve this, *PaPaRa* uses a dynamic programming approach similar to [NW70], described in detail in Section 2.3, which iteratively fills a scoring table H with the ancestral sequence L, as column labels and the query sequence V, as row labels. Additionally, it uses a mechanism (from [Got82]) that encourages sequential gaps in contrast to isolated gaps. This reflects the biological processes more accurately, since one mutation can produce multiple gaps. The gap penalty, that distinguishes between inserting a beginning gap (*open*) and a following gap (*extend*) is called *affine*.

In addition to smaller penalties for sequential gaps, conditional penalties based on the gap flag vector are introduced. The resulting recursion step, taken from the *PaPaRa* V.2 paper [BS12], looks like this:

$$\begin{aligned}
CG^j &= \begin{cases} -3 & \text{if gap on site } j \\ 0 & \text{otherwise} \end{cases} \\
(GP_O, GP_E) &= (-3, -1) \\
(GP_O^j, GP_E^j) &= \begin{cases} (0,0) & \text{if gap on site } j \\ (GP_O, GP_E) & \text{otherwise} \end{cases} \\
S^{i,j} &= \begin{cases} 2 & \text{if } L^j \text{ and } V^i \text{ match} \\ 0 & \text{otherwise} \end{cases} \\
H_I^{i,j} &= \max \begin{cases} H^{i-1,j} + GP_O + GP_E \\ H_I^{i-1,j} + GP_E \end{cases} \\
H_D^{i,j} &= \max \begin{cases} H^{i,j-1} + GP_O^j + GP_E^j \\ H_D^{i,j-1} + GP_E^j \end{cases} \\
H^{i,j} &= \max \begin{cases} H_I^{i,j} \\ H_D^{i,j} \\ H^{i-1,j-1} + S^{i,j} + CG^j \end{cases}
\end{aligned} \tag{3.7}$$

GP_O: gap insertion penalty open
GP_E: gap insertion penalty extend
CG: gap deletion penalty
S: match reward
H_I: insertion scoring matrix
H_D: deletion scoring matrix
H: scoring matrix

In Figure 3.4, the above formulas are illustrated. Penalizing multiple sequential gaps less than each gap on its own is the core idea of [Got82] and leads requires maintaining three scoring matrices: H , H_I and H_D (visualized white, green and blue) instead of only H.

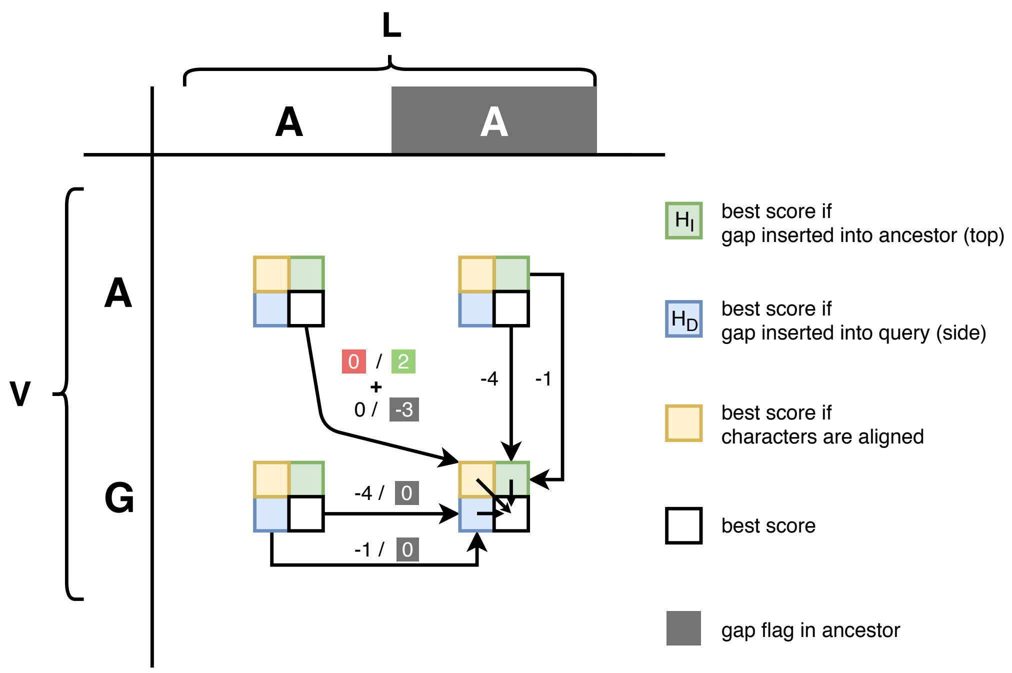
L
A
A
V
A
G
0 / 2
+
0 / -3
-4
-1
-4 / 0
-1 / 0
H_I best score if gap inserted into ancestor (top)
H_D best score if gap inserted into query (side)
best score if characters are aligned
best score
gap flag in ancestor

Figure 3.4: Exemplified *PaPaRa* alignment process. Alignment of segment "AG" of query V to segment "AA" with gap flag vector "01" of ancestor L. The score $H^{2,2}$ (white square) is computed by maximizing $H_I^{2,2}$ (green), $H_D^{2,2}$ (blue) and a temporary value (yellow). The arrows show how they are in turn created from the scores of previous iteration steps. The green value $H_I^{2,2}$ is created from the cell values of above matrices by selecting the maximum of $(H_I^{1,2} - 1)$ and $(H^{1,2} - 4)$. The blue value $H_D^{2,2}$ is the maximization of $(H_D^{2,1} - 0)$ and $(H^{2,1} - 0)$. The value 0 (grey) is chosen over -1 and -4 respectively, because on site 2 of ancestor L the gap flag is set. The temporary value (yellow) is created from the diagonal score $H^{1,1} + 0 - 3$, because $V^2 = G$ and $L^2 = A$ do not match (red) and again the gap flag is set on site 2.

Cell $H_I^{i,j}$ contains the best score if a gap is inserted into the ancestral sequence after character L^j. It can either be the first gap behind L^j, which adds the gap open penalty $GP_O = -4$ to the previous (above) score $H^{i-1,j}$, or a following gap, which in turn adds the extend gap penalty $GP_E = -1$ to the previous (above) insertion score $H_I^{i-1,j}$. Each of those computations is represented by one $\downarrow$ arrow in Figure 3.4 and the larger of the two scores is inserted into the mutual, green target square. This arrow maximization interpretation is applicable to every arrow set that leads to the same square.

Now, the gap flag vector created in the previous chapter comes into play. For each position, it can tell whether aligning a gap or a character is preferable. A set gap flag is visualized with a grey background, for instance behind the second ancestral character $L^2 = $ A, and enables inserting gaps into V at that position with a penalty of zero (also marked grey). This is also described by the conditional gap penalties (GP_O^j, GP_E^j) in the equations. If the gap flag is not set, the deletion score H_D behaves analogously to the insertion score H_I calculation from before. A deletion with respect to the ancestor is equivalent to inserting a gap into the query sequence.

The conditional penalties mentioned in the beginning can, for example, be seen in the diagonal step, where aligning matching characters is rewarded $(2 + 0 = 2)$ if no gap is present, and punished $(2 - 3 = -1)$ otherwise.

It is worth remembering that characters in the ancestral sequence can be ambiguous and therefore multiple characters are allowed to be matched against them even though this has no effect on the way the algorithm works. Additionally, in reality, memory can be saved by storing only one row for H and H_I and only one cell value for H_D, because cell computation requires memory accesses that are at most one cell up and one to the left.

Chapter 4 Vectorization

PaPaRa uses *vectorization* to parallelize the alignment process of Section 3.3. Vectorization refers to the simultaneous execution of multiple basic instructions, such as additions. This is achieved by dedicated CPU dependant instructions which operate on separate registers. Therefore, the code needs to be adapted to use this approach. The *single instruction multiple data* (SIMD) idea is accomplished by performing each instruction on vectors of data instead of a single instance. *PaPaRa* utilizes vectorization by simultaneously and independently aligning one query sequence against multiple ancestors as highlighted in Figure 4.1.

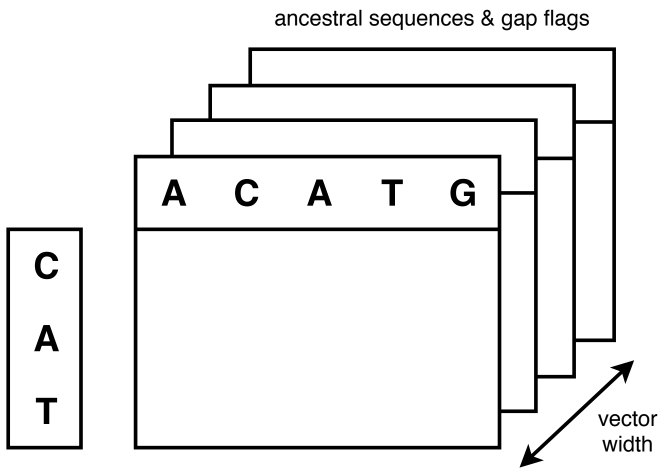

Figure 4.1: Vectorization idea of the *PaPaRa* alignment process. The vector width corresponds to the number of ancestral sequences that the query sequence is aligned to in parallel. This width depends on the hardware.

Instructions	Published	Register Width	GCC flag	Notes
MMX	1997	64 bit	-mmmx	
SSE	1999	128 bit	-msse	
SSE2	2000	128 bit	-msse2	added 144 instructions
SSE3	2004	128 bit	-msse3	added horizontal register instructions
SSSE3	2006	128 bit	-mssse3	added packed integer instructions
SSE4a	2007	128 bit	-msse4a	only supported by AMD CPUs
SSE4.1	2007	128 bit	-msse4.1	added 47 instructions
SSE4.2	2007	128 bit	-msse4.2	added text processing instructions
AVX	2011	256 bit	-mavx	now uses 256 bit registers
AVX2	2012	256 bit	-mavx2	
AVX512	2016	512 bit	-mavx512f -mavx512bw	

Table 4.1: Summary of the Vector Instruction Sets used by *PaPaRa*. The flag column defines the flag that needs to be set when compiling with GCC to enable the use of the specified instructions.

Section 4.1 Technical Background

The used vector instructions depend on the CPU's manufacturer. *PaPaRa* supports the following instruction sets:

The C++ header file *immintrin.h* specifies so called *intrinsic functions* that enables calls to the processor instructions. Once the header file is included, for most compilers (for instance GCC) the intrinsic functions also require specifying the corresponding flag at compile time, see Table 4.1.

Section 4.2 Implementation

The original *PaPaRa* vectorization encapsulates the access to the processor specific instructions inside a templated `vector_unit`. This templated structure `vector_unit` is instantiated for any hardware dependant instruction set, listed in Table 4.1 and therefore allows for the use of the same interface, even though different hardware is operated each time.

The template parameter is the *vector width* that determines how many elements of a given datatype the vector can contain. For example, a nucleotide of a DNA sequence is represented by a *short integer* of 16 bit and thus 32 shorts can be stored within an *AVX 512* register. *PaPaRa* automatically selects the supported vectorization that has the largest vector width. The appropriate code is selected with the help of pre-processor directives which are set during configuration.

In this thesis, as part of the *PaPaRa* optimization, a new instance of `vector_unit` is created that uses the *AVX 512* instruction set. It doubles the register size from 256 bit to 512 bit. The resulting implementation is mostly analogous to existing ones, see *PaPaRa*'s code [Ber16]. The alignment algorithm of *PaPaRa* uses the abstract `vector_unit` operations and can therefore directly work with the new instance. The adaptations are exemplified with the `load` function, which takes an address pointer and loads 512 bit of data into a processor register.

```
static inline const vec_t load( const T* addr ) {
    return _mm512_load_si512(addr);
}
```

The return type `vec_t` is also templated and can be used in operational functions such as the following addition. Here both registers are interpreted to contain a series of short integers that are individually added, when adding the register values.

```
static inline const vec_t add( const vec_t &a, const vec_t &b )
   {
    return _mm512_add_epi16 ( a, b );
}
```

When using a different instruction set the signature does not change, but the implementation does. Seen here for the SSE implementation:

```
static inline const vec_t add( const vec_t &a, const vec_t &b )
   {
    return _mm_add_epi16( a, b );
}
```

The AVX 512 implementation requires the *AVX-512 Byte and Word* extension for the support of 16 bit integers.

Section 4.3 Results

The runtime improvements are analyzed in Table 4.2 with 64 threads and in Table 4.3 for 12 threads. The three data sets used are discussed further in Section 7.3.1. The savings achieved by the *AVX 512* instructions compared to *AVX 256* is consistently above 25%.

Data set	Ancestors	Instructions	Runtime	Improvement
Neotrop	1021 x 4686 bp	AVX 256	276 s	
		AVX 512	211 s	26% faster
Tara	7493 x 3374 bp	AVX 256	713 s	
		AVX 512	427 s	41% faster
BV	1591 x 2763 bp	AVX 256	172 s	
		AVX 512	106 s	39% faster

Table 4.2: Runtime Analysis of Vectorization. Aligned 1000 queries per data set with 64 threads.

When using a more standard setup with 12 threads the improvement is around 40% for all data sets.

Data set	Ancestors	Instructions	Runtime	Improvement
Neotrop	1021 x 4686 bp	AVX 256	3343 s	
		AVX 512	1898 s	44% faster
Tara	7493 x 3374 bp	AVX 256	7967 s	
		AVX 512	4512 s	44% faster
BV	1591 x 2763 bp	AVX 256	1929 s	
		AVX 512	1136 s	48% faster

Table 4.3: Runtime Analysis of Vectorization. Aligned 1000 queries per data set with 12 threads.

Chapter 5 K-mer Heuristic

To restrict the amount of sequence pair-wise alignments *PaPaRa* has to conduct, an efficient heuristic approach is desirable. Here, the idea is to use means that rate similarity between the ancestral DNA and the meta-genome DNA under investigation, without having to compute an expensive alignment - these methods are called *alignment-free*. They are advantageous, because they generally require time that is linear in the sequence length.
A large subcategory of alignment-free methods are *k-mer based approaches*[Sim+09; BSB13; Zie+17]. They divide a larger DNA sequence into DNA fragments and then store a distribution of the fragment occurrence numbers, also known as fragment frequencies, into vectors. The fragments are of fixed length k and are usually called *k-mers*. Analogously, the corresponding frequency vectors are called *k-mer frequency vectors*.

The next section describes how these are created systematically from a given DNA sequence and adjusted for use in *PaPaRa*. To complete the heuristic, a function for sequence comparison (or in this case comparing the corresponding vectors) is needed. With its help, similarity can then be quantified in terms of "distance", which measures the "closeness" between pairs of k-mer frequency vectors. Section 5.2 introduces different distance functions and discusses their uses and advantages from a theoretical perspective.

Section 5.1 K-mer Frequency Vector Creation

In order to create a k-mer frequency vector for a given DNA sequence, the algorithm iterates over the sequence one character at a time and collects every substring of length k. Every k-mers' occurrence is counted and stored in the entries of the corresponding k-mer frequency vector. If a k-mer is not contained in the sequence, the corresponding vector entry is set to zero. Each k-mer is assigned a unique index in the vector based on its position in lexicographic order. This creation process is exemplified in Figure 5.1 with the sequence "AAGTAAG": It contains five k-mers of length three (3-mers): "AAG", "AGT", "GTA", "TAA", and "AAG". The frequency vector lists how often each 3-mer appears, as shown on the right.

Through this translation process, all information on the k-mers' location is ignored. Only a small notion of sequentiality is conserved because of the k-mer overlap.

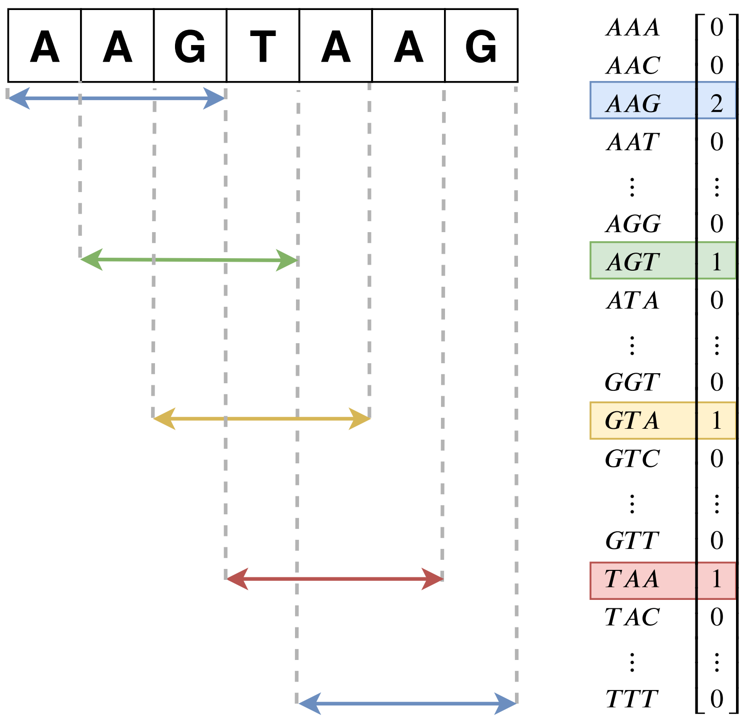

Figure 5.1: k-mer frequency vector creation. For the given DNA sequence "AAGTAAG" the 3-mer frequency vector is created by iterating over every position in the sequence.

Subsection 5.1.1 Quantifying Ambiguity

The k-mer frequency vector creation appears straight-forward at first glance, but an additional layer of complexity is introduced by the ancestral sequences created by *PaPaRa* (see Chapter 3). They can contain *ambiguity characters* that are placeholders for a set of nucleotides. For example, the character M means the corresponding DNA site can contain either an A or a C.

Each k-mer containing ambiguous characters represents multiple possible k-mers. The number of possible k-mers represented by the ambiguous characters is denoted by l. Then the amount, each of these l k-mers contributes to its respective position in the frequency vector, is defined as $1/l$. The idea is to attach less weight to the position of ambiguous k-mers than that of unambiguous ones. This way, when evaluating the closeness to a query, an ancestor that contains an exact match of the query k-mer is explicitly preferred over an inexact match.
In the following example Figure 5.2 the 3-mer "MMG" stands for the 3-mers "AAG", "ACG", "CAG" and "CCG". As a result, each of their frequency vector entries is incremented by $1/4$.

Subsection 5.1.2 Alphabet Reduction

The results of [HR07] suggest that there are cases where translating nucleotide characters into a smaller alphabet and performing phylogeny reconstruction on them

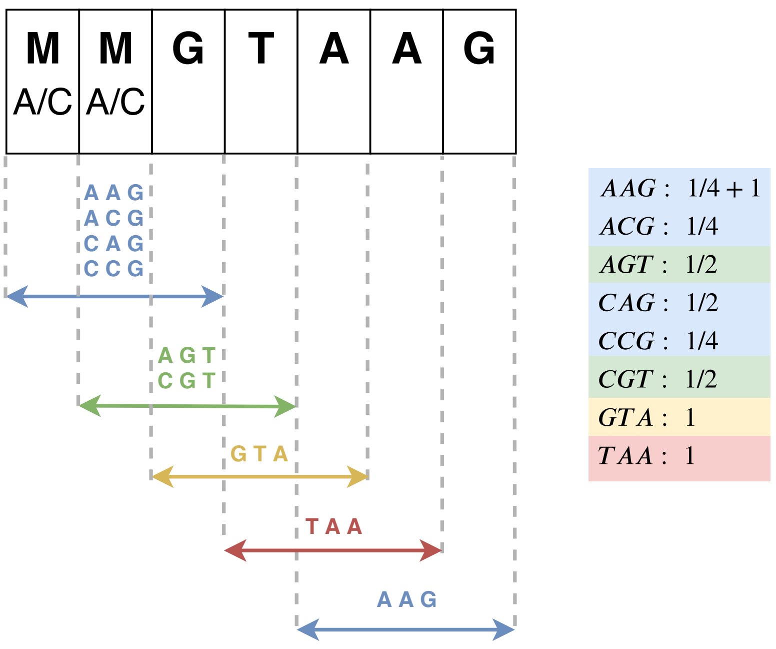

Figure 5.2: k-mer frequency vector creation for sequences containing ambiguity characters. For the given DNA sequence “MMGTAAG”, the 3-mer frequency vector is created. Ambiguous k-mers have less weight.

can yield similar or even better phylogenies on some data sets, even though the smaller alphabet contains less information. A main advantage of this is that the additional complexity added through ambiguous characters in the ancestral sequences can be reduced and thereby runtime improvements can be achieved.

The DNA alphabet $\{A, C, G, T\}$ is reduced to only two characters $\{0, 1\}$ with A and G corresponding to 0 and C and T becoming 1. This assignment is based on the chemical similarity of these pairs [CPM18]. The complete substitution mapping is shown in Table 5.1. The unique bases are marked grey. The rest consists of possible subsets of them, represented as ambiguous characters. Ambiguous characters are converted by majority rule, if the set contains more 0-characters it becomes 0 and vice versa. In case of a tie, it turns into the ambiguous character 2 representing 0 or 1.

old alphabet	A	C	M	G	R	S	V	T	W	Y	H	K	D	B	*
nucletides	A	C	A C	G	G A	G C	G A C	T	T A	T C	T A C	T G	T G A	T G C	T G C A
new alphabet	0	1	2	0	0	2	0	1	2	1	1	2	0	1	2

Table 5.1: Alphabet reduction table. The DNA alphabet is reduced to $\{0, 1, 2\}$.

Section 5.2 Distances

Now, with the techniques of Section 5.1, DNA sequences can be translated into k-mer frequency vectors. To quantify the similarity of sequences, their vector representations are evaluated with methods of linear algebra. Metrics are used to measure distances between vectors. The goal is to find a metric in which the proximity of vectors correlates with the similarity of the respective sequences.
PaPaRa assigns a high score to sequences with long, consecutive matches. Therefore, sequences need to have close k-mer frequency vectors in the used metric. Once a reliable approximation is found, the distance between k-mer frequency vectors can be used as an efficient approximation for pre-filtering ancestors in order to minimize costly *PaPaRa* calculations.

Distance metrics are a well studied field. The following four were chosen and tested in the experiments, described in Chapter 8.
The first and most straight forward distance is the *euclidean distance*. The *cosine distance* measures angles between vectors and is often used in text comparisons. The *Jensen-Shannon divergence* uses information theory to compare probability distributions. Finally the *fingerprint distance* was created as part of this thesis to produce easily interpretable results. The following formulas are defined on the vectors a and b of dimension n and a specific entry of index i is accessed via a_i or b_i respectively.

Subsection 5.2.1 Euclidean Distance

The *euclidean distance* is a geometric approach calculating the distance between two points in an n-dimensional space. Equation 5.1 shows how the euclidean distance is computed. A weakness, according to [AHK01], is its behaviour on sparse vectors, because the difference in distance between the farthest and nearest neighbour of a vector tends to become minimal and therefore signal is lost.

$$eucl(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{5.1}$$

Subsection 5.2.2 Cosine Distance

The *cosine distance* (shown in Equation 5.2) determines 1 - the cosine of the angle between two k-mer frequency vectors. This allows for an abstraction from the vector magnitude, which exhibits advantages when comparing sequences of different length. This is the case when short next-generation sequences are analyzed with respect to around 20 times longer reference sequences. Additionally, it has favourable runtime properties since zero entries are ignored. For positive input, it delivers results in $[0, 1]$. The optimal value 0 is not only achieved in the trivial case of $a = b$, but also in the case when the vectors point in the same direction (linearly dependent).

$$\cos(a,b) = 1 - \frac{a \cdot b}{||a||_2||b||_2} = 1 - \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \cdot \sqrt{\sum_{i=1}^{n} b_i^2}} \tag{5.2}$$

Subsection 5.2.3 Jensen-Shannon Divergence

The *Jensen-Shannon Divergence* (*JSD*)[Lin91] is based on the *Kullback-Leibler Divergence* (*KLD*)[KL51], which measures the similarity of two probability distributions. Equation 5.3 elucidates the addition made to the *KLD*: the *JSD* is computed by calculating the *KLD* of both vectors with respect to their mean vector m. This is done to ensures symmetry, which means $JS(a,b) = JS(b,a)$.

Through the *KLD*, the *JSD* is linked to information theory and its methods of measuring the information contained in probability distributions. The frequency vectors a and b are normalized, by adding all entries and then dividing them by their sum. The results are the probability distributions p and q, which are now fit to be analyzed via information theoretic methods.

The authors of [Sim+09] have already applied the Jensen-Shannon Divergence to the reconstruction of phylogenies, which is a similar context as examined in this thesis.

$$\mathit{JSD}(p,q) = \frac{1}{2}\mathit{KLD}(p,m) + \frac{1}{2}\mathit{KLD}(q,m) \tag{5.3}$$

with mean vector

$$m = \frac{p+q}{2}$$

and Kullback-Leibler Divergence

$$\mathit{KLD}(p,m) = \textstyle\sum_{i=1}^{n} p_i \log_2(\frac{p_i}{m_i}).$$

Subsection 5.2.4 Fingerprint Distance

As indicated above, the main purpose of any distance function is to implement an efficient filtering of ancestral sequences for those that are likely to be similar to a query sequence and, therefore, worth a thorough *PaPaRa* based alignment. As a prerequisite for a perfect match, each k-mer of the query must show up in the ancestral sequence as well.

Therefore, the *fingerprint distance* (FPD) only considers the k-mers of the query. It checks if they occur in the ancestor at least as often as in the query. If this is the case for every k-mer, the fingerprint distance function returns zero. If none of the query k-mers occur in the reference, the function returns one. This would mean that the ancestral sequence under investigation can not contain the query sequence.

The algorithm for the fingerprint distance $\mathit{FPD}(q,a)$ between the frequency vector q of the query and the frequency vector a of the ancestor is defined as:

$$\begin{aligned} \mathit{FPD}(q,a) &= \frac{\sum_i fpd(q_i,a_i)}{\sum_i q_i} \\ fpd(q_i,a_i) &= \begin{cases} 0 & \text{if } q_i = 0 \\ q_i - a_i & \text{if } q_i >= a_i \\ 0 & \text{if } a_i > q_i \end{cases} \end{aligned} \tag{5.4}$$

The value is normalized by the total amount of k-mers in the query. All entries of the query's k-mer frequency vector that have the value zero are ignored.

The fingerprint distance quantifies if the query perfectly matches the ancestor:

- Zero : Query sequence may perfectly match the current ancestor.
- Value between 1 and 0: Percentage of query k-mers which certainly do not match to the ancestor.
- One: Query does not match to the current ancestor at all.

The *FPD* function can be compared to a criminological fingerprint investigation: If you, for example, find a complete fingerprint on a weapon, you search for persons with the same exact fingerprint. Maybe, however, the fingerprint on the weapon was not complete, but only partial. Then you search for people, who match all notches of the partial fingerprint. When not all notches match, the *FPD* is above 0. This could be the effect of smudges. When none fit, the *FPD* is 1.

Visual interpretation. The fingerprint distance $FPD(q, a)$ is the percentage of the query k-mers that do not occur in the ancestor.

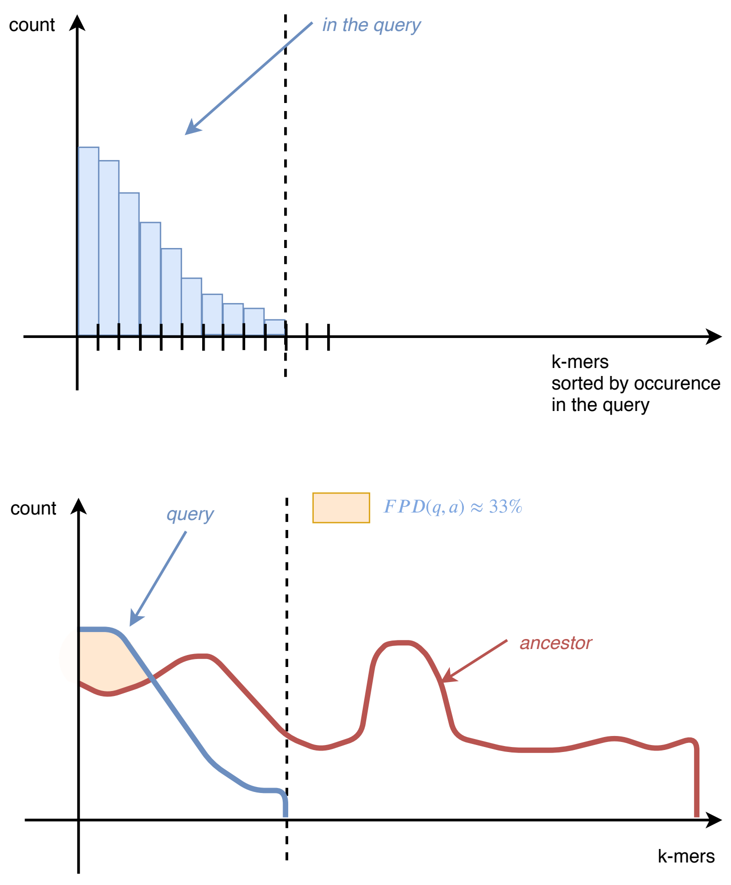

Figure 5.3: Visualization of the fingerprint distance between query and ancestor. The k-mer frequency vectors of the query were rearranged by frequency. The ancestral k-mer entries are compared to this.

Chapter 6 K-mer Heuristic Implementation

The previously described k-mer heuristic was integrated into to *PaPaRa* after the ancestral sequence creation and prior to the time-consuming *PaPaRa* scoring, see Figure 6.1. In there, preexisting *PaPaRa* elements are marked blue and the heuristic extension introduced is marked in green.

The critical implementation details for the heuristic are as follows.
The controlling point for all methods related to k-mer frequencies is the class `kmer_frequencies`. It uses the strategy pattern from [Wik19a] to manage the alternative k-mer frequency creation methods - ambiguity quantification and alphabet reduction - discussed in Section 5.1. The different variants are named *strategies* and have a common signature defined via an interface, realized with an abstract class in C++.

```
class CreateFreqVecStrategyInterface
{
    public:
        virtual unordered_map<int, float> execute(const vector<
            int> &seq, size_t k, size_t num) const = 0;
};
```

This signature shows that the strategies take a sequence and translate it into a k-mer frequency vector.
The datatype for DNA sequences is `vector<int>`. Since the k-mer frequency vectors were expected to be sparse, the implementation uses `unordered_map<int, float>` from the standard library to represent them. It stores (key, value) pairs and supports fast lookup through hashing mechanisms. This way, only non-zero vector entries are stored. Also the vectors belonging to the ancestors are only computed once and are managed by the `kmer_frequencies` object.

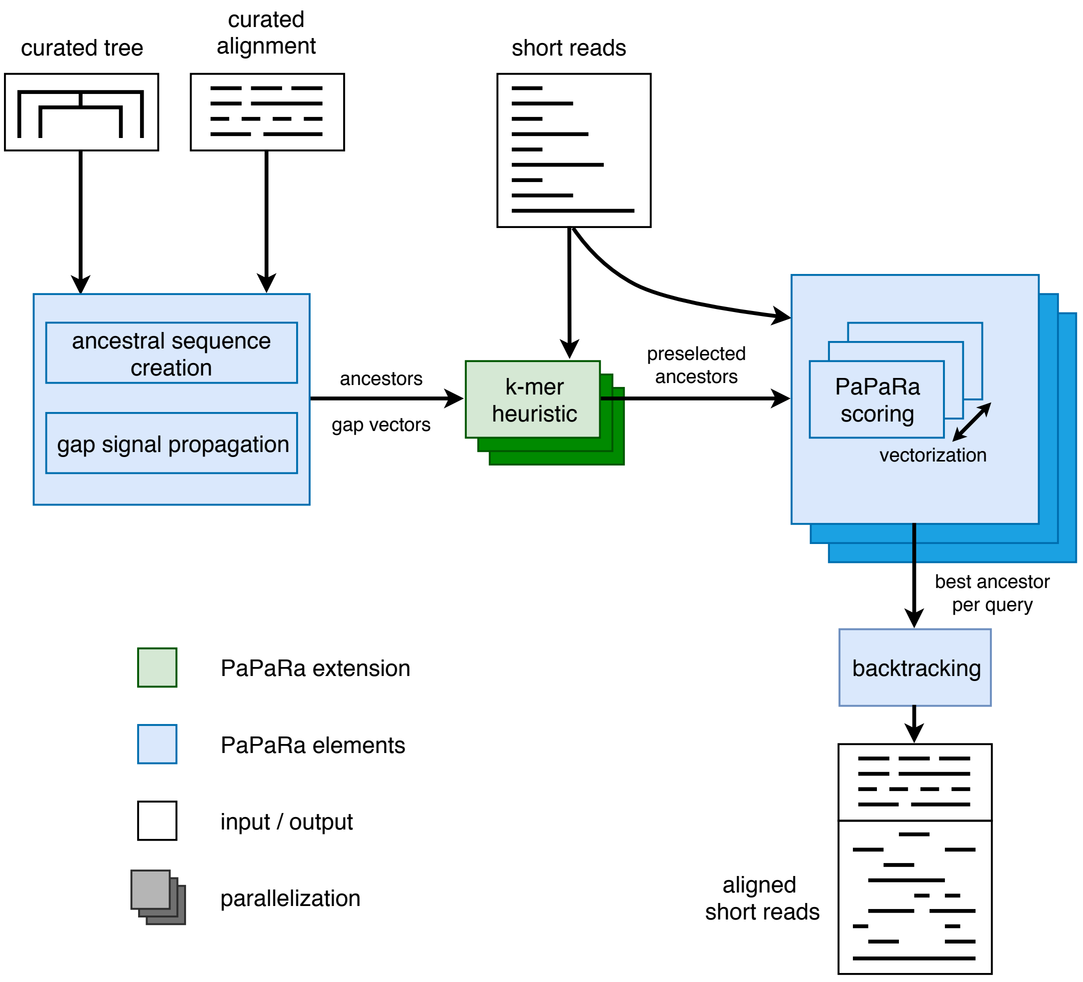

Figure 6.1: Heuristic in context of the *PaPaRa* pipeline. First *PaPaRa* creates the ancestral sequences and gap flag vectors with the help of a curated reference alignment and the associated phylogeny. The k-mer heuristic created in this thesis then filters out ancestors that are deemed unlikely candidates for a high scoring *PaPaRa* alignment. For each of the preselected ancestors, the original *PaPaRa* scoring procedure is performed and the best ancestor is selected and used as a basis to create the actual alignment.

Section 6.1 K-mer Frequency Vector Creation Basics

For the frequency vector creation, the k-mers inside a sequence need to be counted. In this context, a mapping from k-mer to vector has been chosen which is easily computable when indexing all possible k-mers in lexicographical order. The index of k-mer s can be obtained by assigning the characters "A, C, G, T" a rank "0, 1, 2, 3" and then multiplying it with position dependent powers of 4 (see Equation 6.1). For example, the index of 3-mer "GAT" is $index(GAT) = 2 \cdot 4^2 + 0 \cdot 4^1 + 3 \cdot 4^0 = 32 + 0 + 3 = 35$.

$$r(s_i) = \begin{cases} 0 & \text{if character on position } i \text{ is } A \\ 1 & \text{if } s_i = C \\ 2 & \text{if } s_i = G \\ 3 & \text{if } s_i = T \end{cases} \tag{6.1}$$
$$index(s) = \sum_{i=1}^{k} r(s_{k-i}) \cdot 4^{k-i}$$

Section 6.2 Ambiguity Quantification

The class `AmbiguityQuantificationStrategy` inherits from `CreateFreqVecStrategyInterface` and implements the strategy for ambiguity quantification (Section 5.1.1).

For the extraction of the unambiguous k-mers, a sequence is traversed several times. First, each ambiguous character is translated into a set of ranks of the represented characters. For example, the sequence "MMTGAAT" is translated into the set sequence $\{0,1\}\{0,1\}\{3\}\{2\}\{0\}\{0\}\{3\}$.
The next iteration cuts out k sets in each step and processes them with another iterative function. This function rearranges the k sets (each holding one to four elements) into l sequences of k elements. Hereby, all possible sequential combinations of the original sets are created. It is important that the order is preserved, because otherwise the incorrect k-mers are created. In the example, $\{0,1\}\{0,1\}\{3\}$ becomes $\{003, 013, 103, 113\}$.
Now the rank representations can be translated into k-mer indices with Equation 6.1, which are $\{3, 7, 19, 23\}$ in the example. Once the index vector is created, each computed position of the k-mer frequency vector is incremented by $1/l$.

Each position of the sequence is processed $k+1$ times on average. The worst case sequence of length n would consist of only character B, which represents 4 nucleotides. In this case, 4^k indices would be produced for each position, which results in $\mathcal{O}((k+1) \cdot 4^k \cdot n)$ index computations.

Section 6.3 Distances

The various distance functions: euclidean distance, cosine distance, Jensen-Shannon divergence, and fingerprint distance all have the same signature. They take two k-mer frequency vectors and return a floating point value:

```
static float distance(const unordered_map<int, float> &freq_vec1
    , unordered_map<int, float> &freq_vec2)
```

As a convention, they iterate only over the first frequency vector, which therefore has to contain all the indices that shall be considered (see implementation of euclidean distance below).
A trick of C++ is, hereby, that the []-operator will return the value of key i if it exists and insert the pair (i,0) into the `unordered_map`, otherwise. This property is

used in all distance implementations. If only the query k-mers are to be analyzed - as, for instance, required for the fingerprint distance - it is sufficient to simply pass the query vector as the first parameter.

```
static float euclidian(const unordered_map<int, float>& vec1,
   unordered_map<int, float>& vec2){
       float result = 0;
       for (auto it = vec1.begin(); it != vec1.end(); it++)
       {
           float elem1 = (*it).second;
           float elem2 = vec2[(*it).first];
           float dist = elem1 - elem2;
           result += dist * dist;
       }

       return result > 0.0 ? sqrt(result) : 0.0;
   }
```

All distance functions, as well as vector normalization on `unordered_map` vectors, are accumulated in the `vector_utils` class.

Section 6.4 Output for Data Analysis

The query-ancestor distances are stored inside the `scoring_results` class and can be printed into files with the help of `output_freq_vec`. This functionality is only required to analyze the created heuristic and can be omitted in the new *PaPaRa* version.
In the created output file, the row indices correspond to the ancestral indices and the pairwise distances of a query fill a column. The data output is kept as simple as possible to allow for maximum flexibility with respect to downstream data analysis.

Section 6.5 Parallelization

The implementation supports data parallelism in the form of threading for ancestral k-mer frequency vector creation of ancestors, as well as the distance computation. In both cases, a thread-safe `index_vector` is used.
In the first task, the index vector assigns an ancestral sequence for translation to each thread. In the second case, the threads are assigned a query for which the thread creates pairwise distances to all ancestors.

Section 6.6 Unit Tests

To improve correctness, unit tests were created for all distances and methods related to k-mer vector creation. For implementation, the unit test framework of the C++ library *Boost* [Roz] was used.

Chapter 7 Evaluation

PaPaRa's main computational effort is caused by having to calculate all pairwise alignments. The approach of this thesis is to apply a heuristic to filter out unpromising ancestral sequences and thereby reduce the number of alignment operations. In this chapter, the performance of the heuristics is measured by analyzing their correlation to the *PaPaRa* scores. More precisely, the applicability of k-mer-frequency vector based heuristics is analyzed.

For the heuristic approach in this thesis, each sequence is translated into a vector of dimension 4^k which, for each possible fragment of length k (k-mer), records how often it appeared in the sequence. Then, the similarity between the query and ancestral sequence is assessed by calculating the distance between these vectors (see Chapter 5.2). The lower it is, the more alike they are and the closer is the assumed evolutionary relationship of the associated species.
In contrast, in the *PaPaRa* approach the score is higher, if the ancestor better matches the query.

To answer the question how well a distance function approximates the *PaPaRa* score, an *approximation plot* as in Figure 7.1 is used. It describes the alignment evaluations of a single query to every ancestor, that is the similarity of the query to every ancestor according to the distance function and the *PaPaRa* score.
Therefore, each of the points represents an ancestral sequence with the respective *PaPaRa* score on the x-axis and the distance function on the y-axis. In the case of the *neotrop* data set, the plot contains 1021 points.
The blue continuous line shown in the plot is the result of linear regression and gives an indication of the relationship between both measures. A perfect linear correlation exists if the trajectory leads downwards and all points are positioned on the line. The correlation in Figure 7.1 is not optimal, but usually lower *PaPaRa* scores do have a higher distance value as well. There are some outliers, indicating that the respective ancestors are very close according to the distance function but attain only a mediocre *PaPaRa* score.
The goal of the algorithm is to quickly find the best *PaPaRa* scoring ancestor p_q for a given query q, by only considering the ancestors with the shortest distances. The best ancestor is represented by the rightmost point, marked with p_7, for the plot of

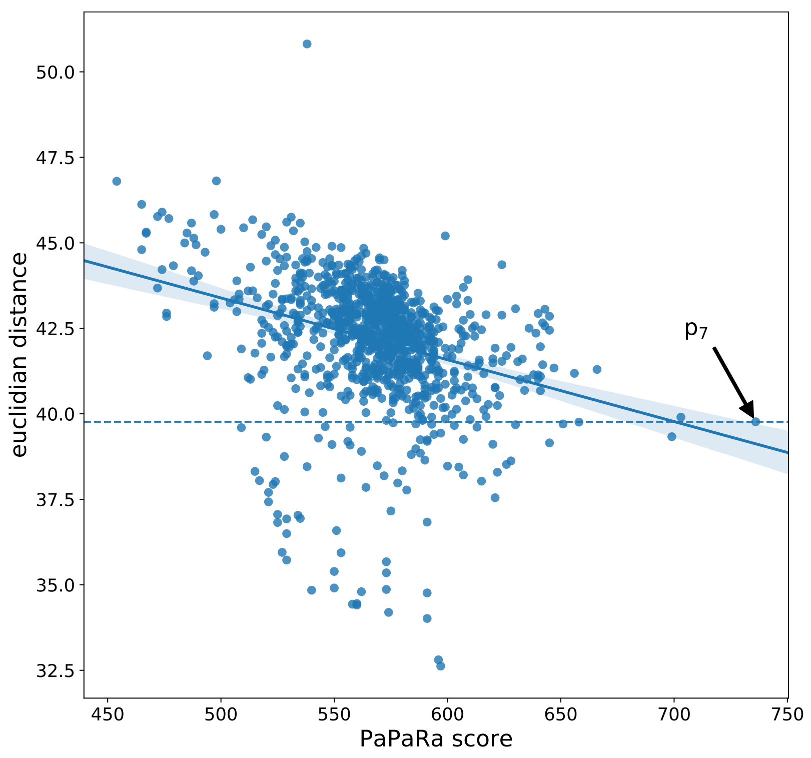

Figure 7.1: Euclidean approximation plot of query 007. The y-axis shows the 7-mer euclidean distance and the x-axis the *PaPaRa* scores for every ancestor towards query 007 for the neotrop datastet. The blue continuous line is the result of a linear regression on the points. The dashed line marks the points that need to be evaluated before the best ancestor p_7 is found. Here, the 7-mer euclidean distance is not optimal, but captures the correct tendency.

query 007. Its distance is also marked via the blue horizontal dashed threshold. The amount of points below that line indicates how many points need to be considered to guarantee that p_q is found.

Section 7.1 Formalization

The following notations are introduced to abbreviate frequently used terms:

q	a query
p_q	the ancestor that achieves the best *PaPaRa* score for query q
r	radius
$U_r(q)$	*neighbourhood of query q* contains all ancestors with distances smaller than the radius r to a query q with respect to a specific distance function
$d(a, b)$	*distance of two vectors a and b* with $d \in \{$euclidean, cosine, Jensen-Shannon, fingerprint$\}$
$U_{d(q,p_q)}(q)$	all ancestors with a lower distance d to q than p_q
$f_d(q) := \frac{\#U_{d(q,p_q)}(q)}{\#\text{ancestors}}$	percentage of ancestors closer to q than p_q, according to the distance function d

Section 7.2 Quality Measures

The analysis was performed with the help of the following three methods.

Subsection 7.2.1 Approximation Plot

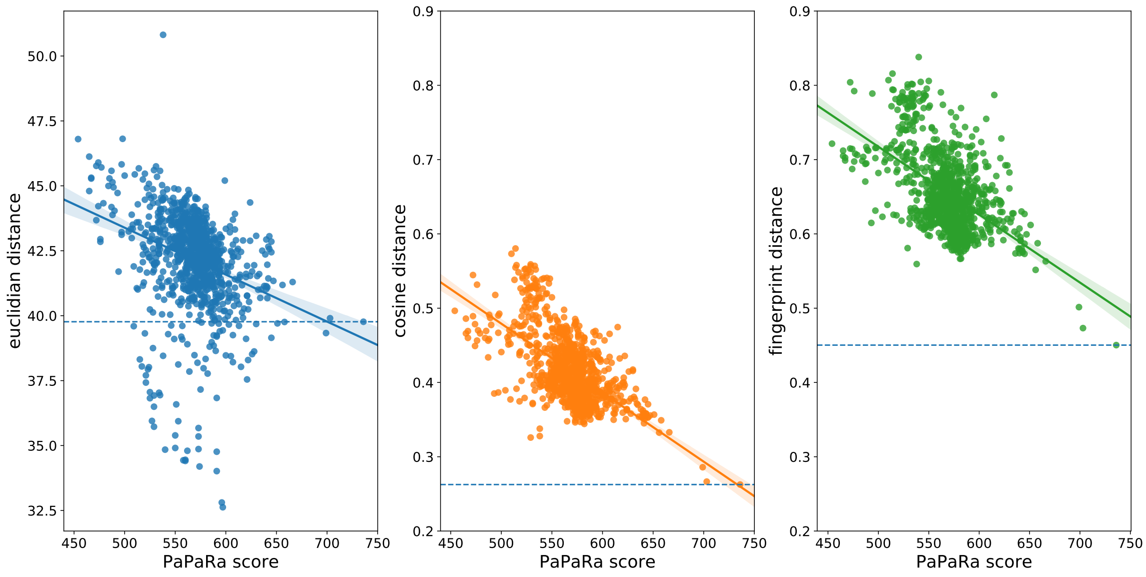

Figure 7.2: Comparison of the euclidean, cosine and fingerprint distance approximation plots. Approximation of the *PaPaRa* score by 7-mer distance functions on the neotrop data set for query 007.

The approximation plot can vary substantially with the used distance metric, see Figure 7.2. The best ancestor p_7 according to *PaPaRa* is also the closest ancestor with the cosine and the fingerprint distance and is therefore found directly for them, while the euclidean distance classifies about 8% of the ancestors as being closer. But it can also be the other way around as shown in Figure 7.3, where the fingerprint distance identifies p_{13} as the worst of all ancestors.

Caution is required, when comparing plots belonging to different queries. The mere values do not contain any information by themselves. It is only relevant how they are classified in comparison to the other points. This is a result of the queries' irregularity in length and nucleotide composition.

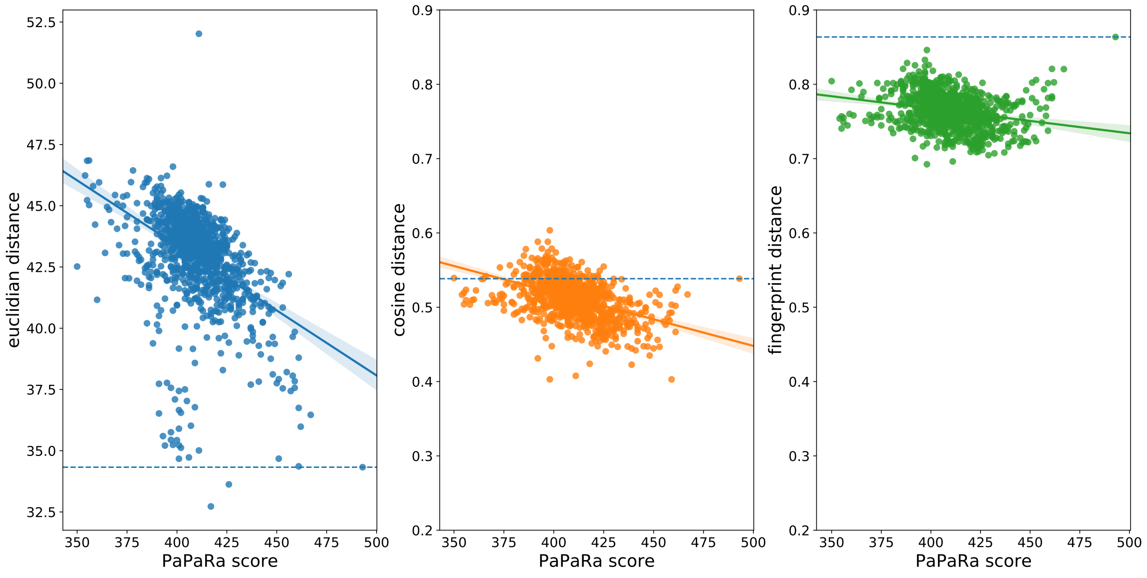

Figure 7.3: Comparison of the euclidean, cosine and fingerprint distance approximation plots. Approximation of the *PaPaRa* score by 7-mer distance functions on the neotrop data set for query 013.

Subsection 7.2.2 Distance Quality

distance	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
Euclidean	6.8	0.1	61.12	7.17	14.79	23.31	26.15
Cosine	10.89	0.1	99.8	25.64	55.24	96.08	98.73
Jensen-Shannon	18.02	0.1	100.0	29.7	82.96	99.9	99.9
Fingerprint	21.18	0.1	100.0	31.68	87.56	99.9	100.0

Table 7.1: Efficiency properties per distance function. Distributions of $f_d(q)$ among all queries in the different 7-mer distance functions on the neotrop data set.

Table 7.1 describes distribution properties of $f_d(q) = \frac{\#U_{d(q,p_q)}(q)}{\#\text{ancestors}}$, the percentage of ancestors closer to the query q than the highest *PaPaRa* scoring ancestor p_q, or visually the percentage of points below the dotted line in approximation plots (e.g. Figure 7.1). The descriptive properties are:

mean	arithmetic average of $f_d(q)$ among all queries q
min	minimum value
max	maximum value
stddev	standard deviation
90% (95%, 98%)	required percentage of ancestors to pass screening, such that for 90% (95%, 98%) of all queries q, the respective p_q is contained

Each row of these tables is visualized via a *histogram* in the following section.

Subsection 7.2.3 Distance Efficiency Histogram

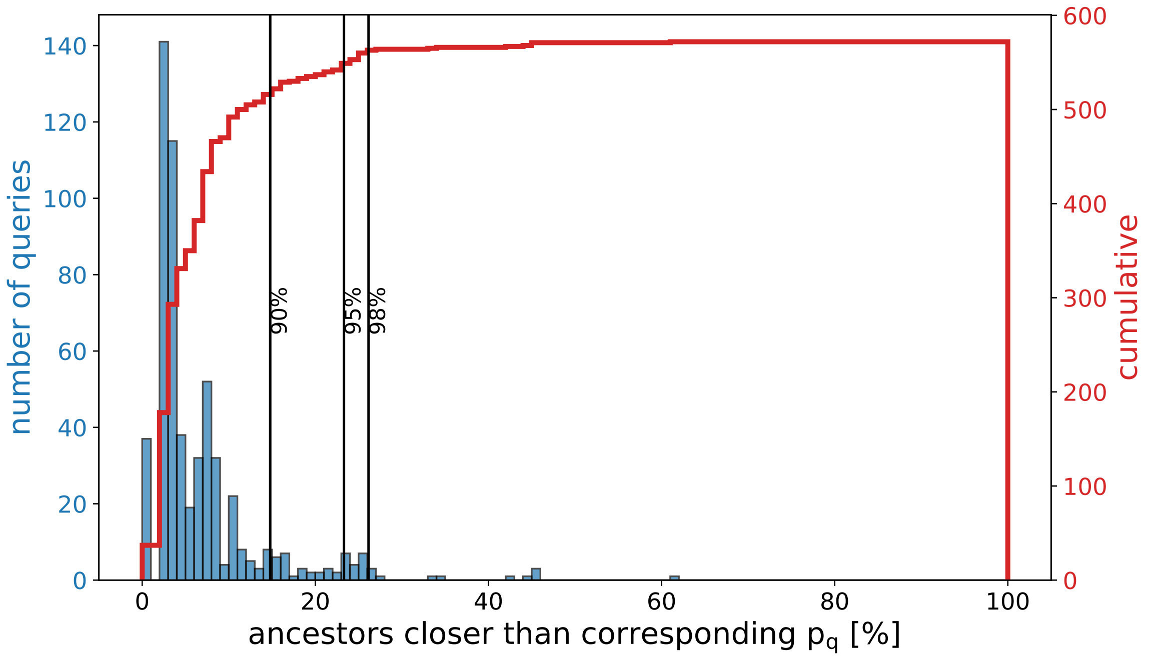

Figure 7.4: Efficiency of euclidean 7-mer distance.
data set: neotrop

To assess the quality of a specific heuristic setting, a histogram plot, as in Figure 7.4 is helpful.
This plot assigns all queries into 100 bins of width 1%, based on the percentage of ancestors that lie below the p_q-threshold. In the example, for around 140 queries, the euclidean 7-mer distance categorizes $2 - 3\%$ of the ancestors as a too close match. Then, these bars are added up to create a cumulative histogram, shown as a red line. Now, if *PaPaRa* is given a percentage (x-axis) of closest ancestors to analyze, then the corresponding cumulative y-value will state how often p_q can be found.
The 90% (95%, 98%) thresholds in Table 7.1 are visualized as vertical lines. So, in the example, the heuristic works in 90% of all cases if 15% of ancestors are more closely examined. This means that 85% of *PaPaRa* alignment runtime can be saved. Subsequently, the influence of k, the distance function and the data set is evaluated.

Section 7.3 Parameters

The following properties were varied among the experiments:

data set	$\in \{tara, bv, neotrop\}$
k	$\in \{3, 4, 5, 6, 7, 8, 9, 10\}$ Denotes the k-mer length. The size of the k-mer vectors increases exponentially with rising k.
distance	$\in \{euclidean, cosine, Jensen\text{-}Shannon, fingerprint\}$ Chooses one of the described distance functions.
remove gaps	$\in \{yes, no\}$ When the removal setting is chosen, the ancestor characters that are on positions with set gap flag are removed. This leads to shorter sequences and inhomogeneous lengths.
reduced alphabet	$\in \{yes, no\}$ Enables the alphabet reduction strategy.
only query k-mers	$\in \{yes, no\}$ If selected, the ancestor frequency vectors are filtered to contain only entries of k-mers that exist in the query.

The test code was encapsulated in a separate header file, which can be imported and activated through a single line of code inside *PaPaRa*. This header file determines the settings and can perform numerous test runs inducing minimal overhead.

Subsection 7.3.1 Data Sets

The experiments were conducted on three separate data sets:

- ***neotrop***, Neotropical data set [Mah+17]:
  The Neotropical data set was collected from soil samples in neotropical rainforests in Costa Rica, Panama and Ecuador. Samples from tropical environments usually have few known references due to the hyperdiversity at the macro- as well as microbiological level.

  References: 512
  Reference Length: 4686 characters
  Queries: 100.000

- ***tara***, Tara Oceans data set [Sun+15]:
  The "Tara Oceans" expedition has collected immense amounts of metagenomic data from 68 locations representing all main oceanic regions. Analyzing this data has led to a large microbial reference catalog, mostly containing novel sequences.

  References: 3748
  Reference Length: 3374 characters
  Queries: 10.000

- ***BV***, Bacterial Vaginosis data set [Sri+12]:
  For the Bacterial Vaginosis data set, 220 women's vaginal fluid was examined. About half of the women were diagnosed with bacterial vaginosis, using four clinical criteria. Classifying bacterial communities helps to associate certain bacteria with each of the disease's diagnostic criteria.

  References: 797
  Reference Length: 2763 characters
  Queries: 15.060

Section 7.4 Technology

The algorithm performance on the three data sets was evaluated under two different technical settings.

Subsection 7.4.1 Evaluation Software

The evaluation was conducted via Python on a *Jupyter Notebook* [Klu+16]. Data management and manipulation was done with the Python library *Pandas* [McK10] and visualization was performed through the *Seaborn* library [Was+18], which is based on *Matplotlib* [Hun07].

Subsection 7.4.2 Hardware

The experiments were run under two different settings as shown in Table 7.2.

	Standard Use Laptop	Multi-Core Server
Operating system	Ubuntu (18.04.2)	CentOS Linux (7)
Main memory	16 GB	754 GB
Processor	Intel i7-8750H	Intel Xeon Gold 6148
Architecture	Coffee Lake	Skylake (server)
Sockets	1	2
Physical Cores	6	40
Virtual Cores	12	80
Vector Intrinsics	sse, sse2, ssse3, sse4_1, sse4_2, avx, avx2	sse, sse2, ssse3, sse4_1, sse4_2, avx, avx2, avx512f, avx512dq, avx512cd, avx512bw, avx512vl

Table 7.2: Specifications of the used setups.

Chapter 8 Results

With the methodology described in Chapter 7, the quality of the heuristics is assessed separately on the three data sets. The differences in the data sets might lead to varying performance.

Section 8.1 Neotropical Data Set

Distances	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
Euclidean	46.4	0.1	96.96	21.55	65.52	70.23	76.79
Cosine	22.48	0.1	94.32	23.28	68.66	75.12	82.76
Jensen-Shannon	20.92	0.1	98.73	24.79	68.17	75.22	86.78
Fingerprint	18.52	0.1	98.43	26.47	72.18	78.06	88.44

Table 8.1: Efficiency properties per distance function. Distributions of $f_d(q)$ among all queries in the different 7-mer distance functions on the neotrop data set.

The first experiments conducted yielded similar results for all distance functions (see Table 8.1).
These heuristics are generally not promising since in all cases only around 30% of the ancestors can be filtered out and, therefore, around 70% of all ancestors still need to be examined with *PaPaRa* (see 90%, 95% and 98% thresholds). Comparing the percentage thresholds of the euclidean distance and the Jensen-Shannon distance ($65\% < 68\%$, $70\% < 75\%$), the euclidean distance seems preferable (see 90% and 95%), but the histograms suggest otherwise. The Jensen-Shannon histogram in Figure 8.2 is closer to an ideal histogram, which has only one peak at 0%, while the euclidean distance histogram in Figure 8.1 does not indicate a correlation.
Interestingly, for every distance there is at least one query where almost all ancestors are estimated to be better than the actual highest *PaPaRa* scoring ancestor p_q (see max).

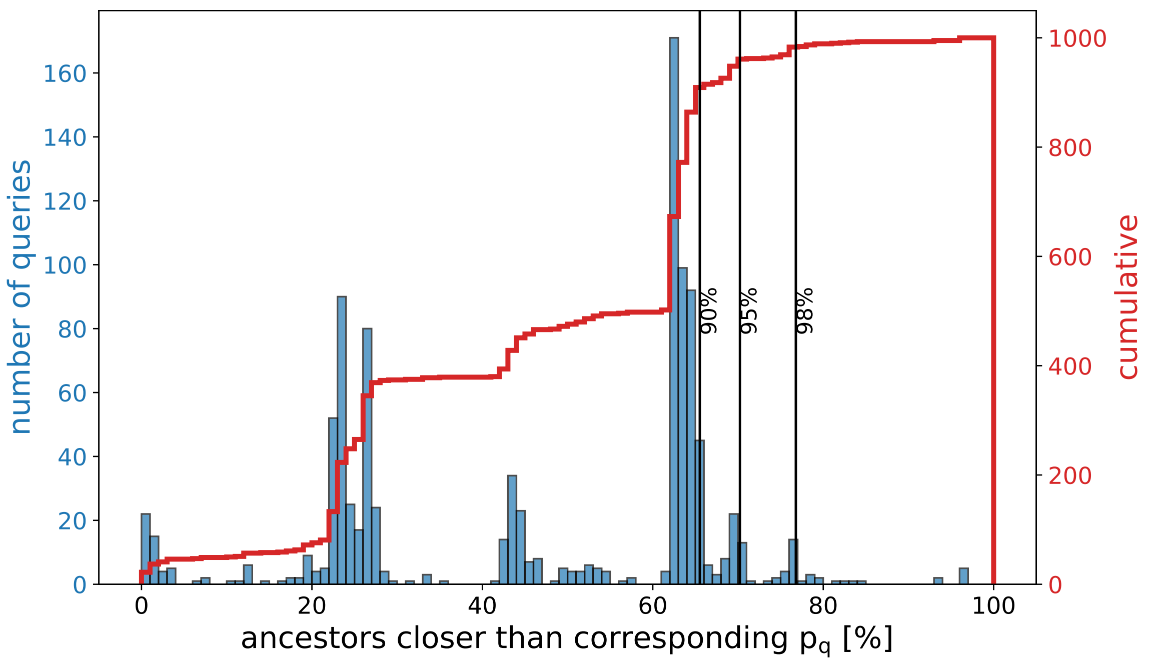

Figure 8.1: Efficiency of euclidean 7-mer distance.
data set: neotrop

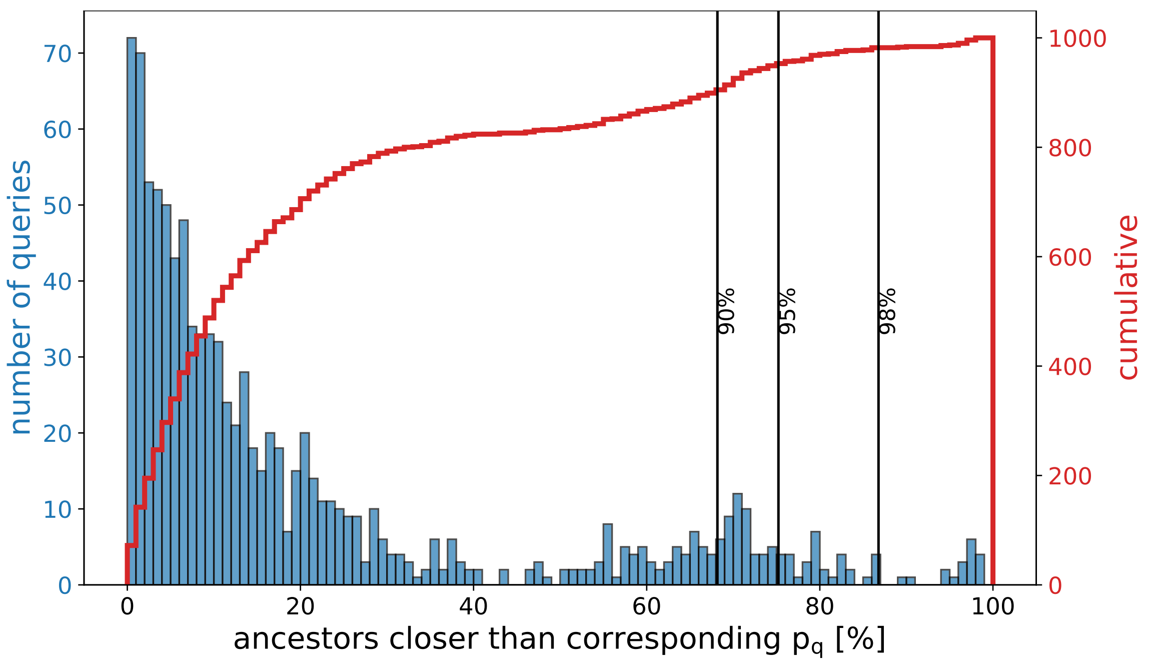

Figure 8.2: Efficiency of Jensen-Shannon 7-mer distance.
data set: neotrop

Subsection 8.1.1 Gap Vector Incorporation

PaPaRa prepares gap flag vectors to determine sites, where a gap is more probable than no gap. During the alignment process, gap insertions into these positions are favored, rather than characters. To account for this feature, the positions where a gap is probable are removed from the ancestral sequences before creating the k-mer vector. Table 8.2 shows that this setting leads to a better approximation of *PaPaRa* scores for all heuristics. The best performance is reached with the cosine distance, as shown in Figure 8.3.

Distances	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
Euclidean	17.69	0.2	52.69	11.45	28.4	39.57	46.23
Cosine	6.01	0.1	99.71	11.12	12.93	17.04	32.81
Jensen-Shannon	8.37	0.1	99.9	14.54	25.95	33.89	52.6
Fingerprint	16.07	0.1	100.0	25.55	66.41	72.38	86.19

Table 8.2: Efficiency properties per distance function. Distributions of $f_d(q)$ among all queries in the different 7-mer distance functions on the neotrop data set. Gaps were removed from ancestors.

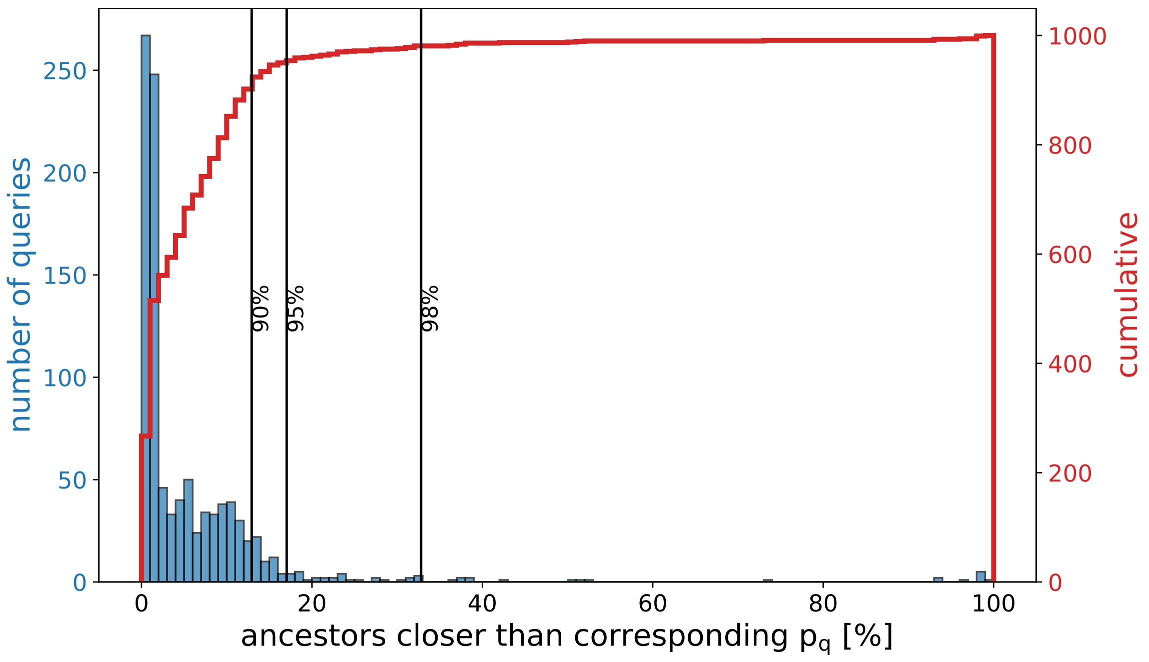

Figure 8.3: Efficiency of cosine 7-mer distance.
data set: neotrop
gaps removed from ancestors

Subsection 8.1.2 Query Reduction

The neotrop data set contains a high amount of unknown species [Mah+17] that can not be analyzed with high confidence. Hence, the queries are filtered to only contain confident queries for which the subsequent phylogenetic placement with EPA-ng [Bar+18] reaches "good" results (meaning they have a likelihood weight ratio [MKA10] of at least 0.95). Table 8.3 shows that the euclidean distance performs significantly better on these filtered queries, while all other metrics yield inferior results.

Distances	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
Euclidean	6.8	0.1	61.12	7.17	14.79	23.31	26.15
Cosine	10.89	0.1	99.8	25.64	55.24	96.08	98.73
Jensen-Shannon	12.03	0.1	99.9	26.43	58.57	98.33	99.51
Fingerprint	21.18	0.1	100.0	31.68	87.56	99.9	100.0

Table 8.3: Efficiency properties per distance function. Distributions of $f_d(q)$ among all queries in the different 7-mer distance functions on the neotrop data set. Gaps were removed from ancestors. Filtered to contain 573 confident queries.

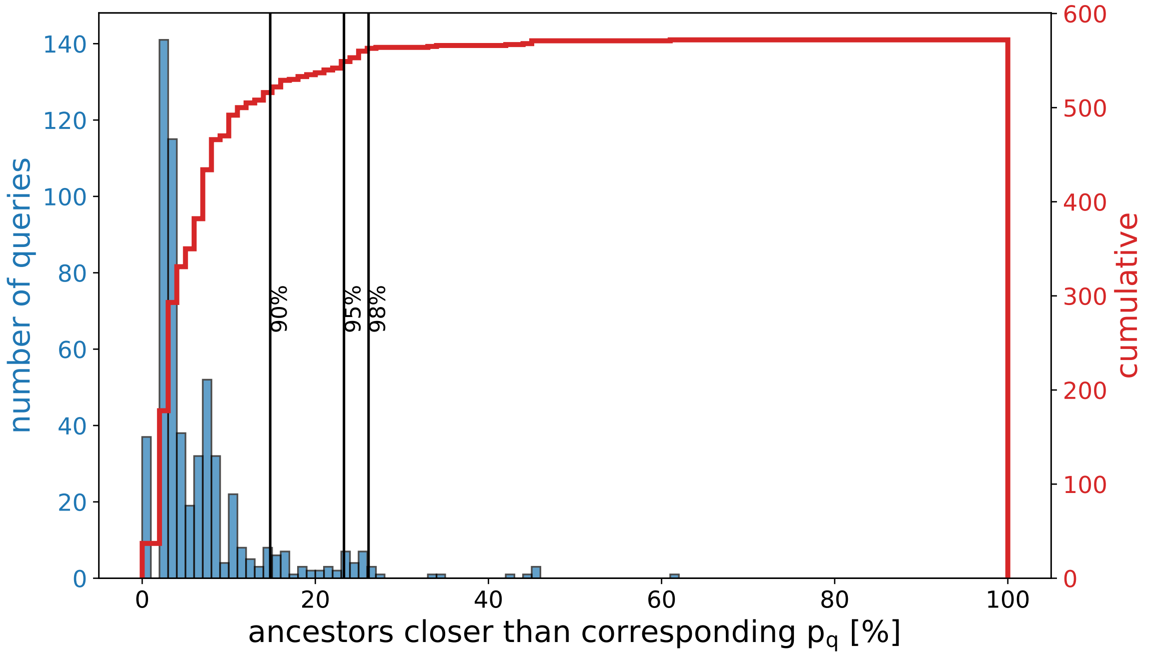

Figure 8.4: Efficiency of euclidean 7-mer distance.
data set: neotrop
gaps removed from ancestors
573 confident queries

When investigating the reason, the cosine histogram (Figure 8.5) shows that the cosine still performs well in most cases, but there are also 5% of queries on the right side of the plot, which distort the thresholds.
One of those queries (query 008) is shown in the approximation plot in Figure 8.6. Here, p_8 is an outlier for the cosine and fingerprint distance. The fingerprint distance

of around 0.86 means that 86% of the query k-mers are not found in the sequence belonging to p_8. This could, for example, mean that gaps are alternating with characters in the aligned query sequence, such that most original k-mers in the query sequence are not present in the aligned query anymore. It remains to be determined, how often this occurs in other data sets.

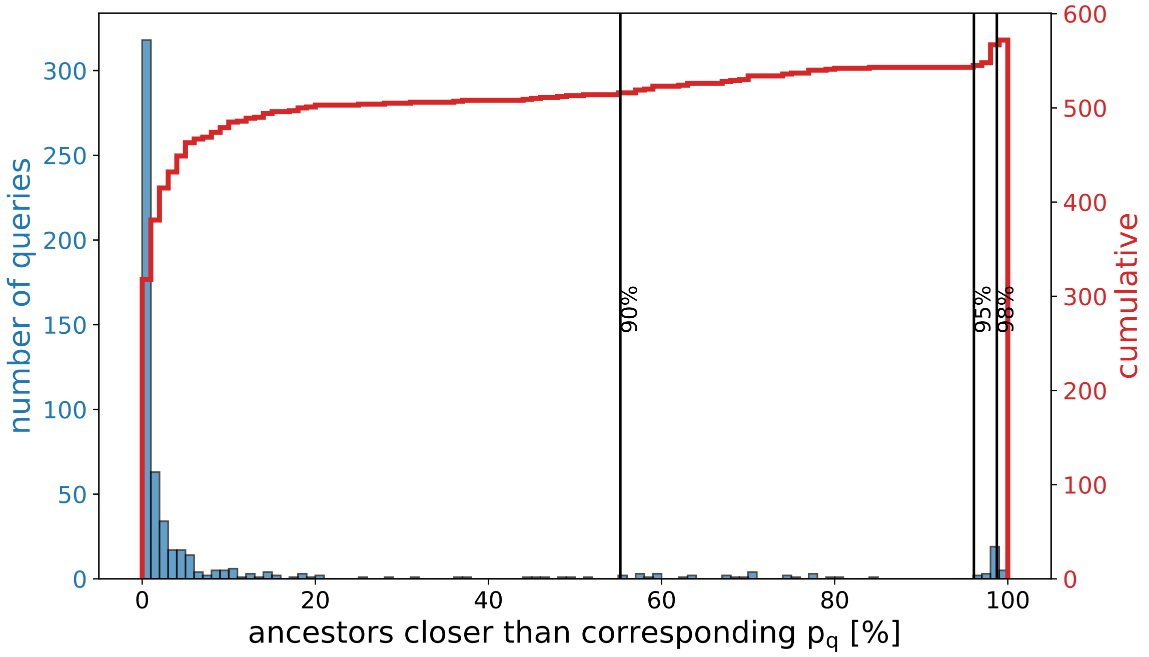

Figure 8.5: Efficiency of cosine 7-mer distance.
data set: neotrop
gaps removed from ancestors
573 confident queries

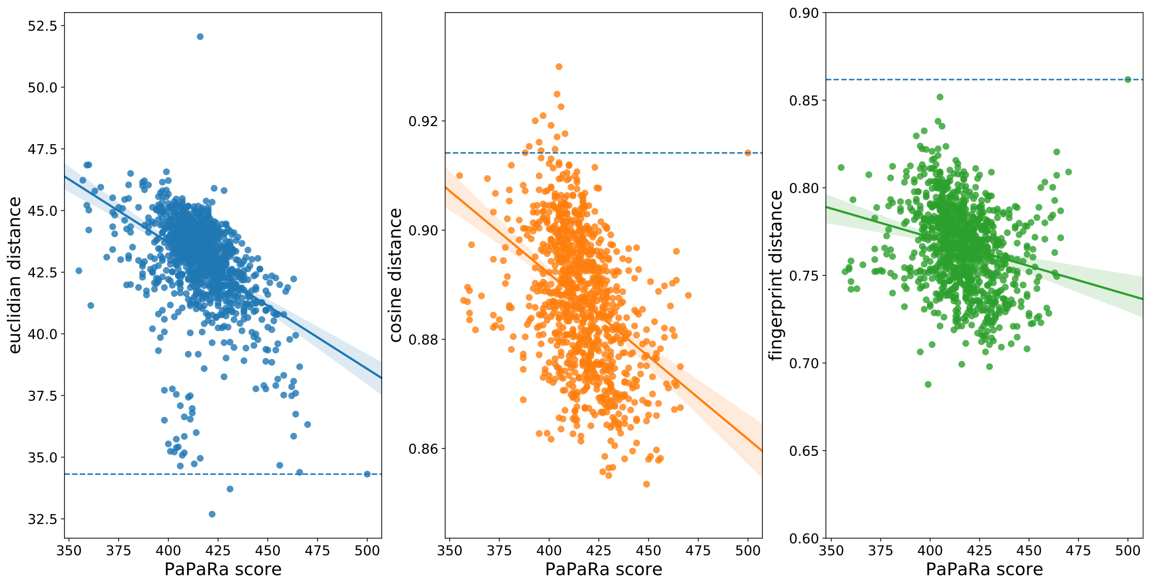

Figure 8.6: Comparison of the euclidean, cosine and fingerprint distance approximation plots. Approximation of the *PaPaRa* score by 7-mer distance functions on the neotrop data set for query 008.

Section 8.2 Tara Data Set

All heuristics deliver good results on the tara data set, see Table 8.4. If a failure rate of 10% is acceptable, all heuristics rule out at least 88% of the ancestors (see 90%). For a more accurate result (98% success rate) the euclidean distance performs significantly better than the other heuristics. An example query distribution is shown in Figure 8.7.

Distances	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
Euclidean	4.1	0.01	99.63	7.24	11.25	16.07	26.25
Cosine	2.79	0.01	99.01	11.21	4.84	13.24	41.23
Jensen-Shannon	4.02	0.01	98.93	13.84	7.71	20.89	64.58
Fingerprint	5.17	0.01	99.39	15.92	10.86	36.15	75.79

Table 8.4: Efficiency properties per distance function. Distributions of $f_d(q)$ among all queries in the different 7-mer distance functions on the tara data set. Gaps were removed from the ancestors.

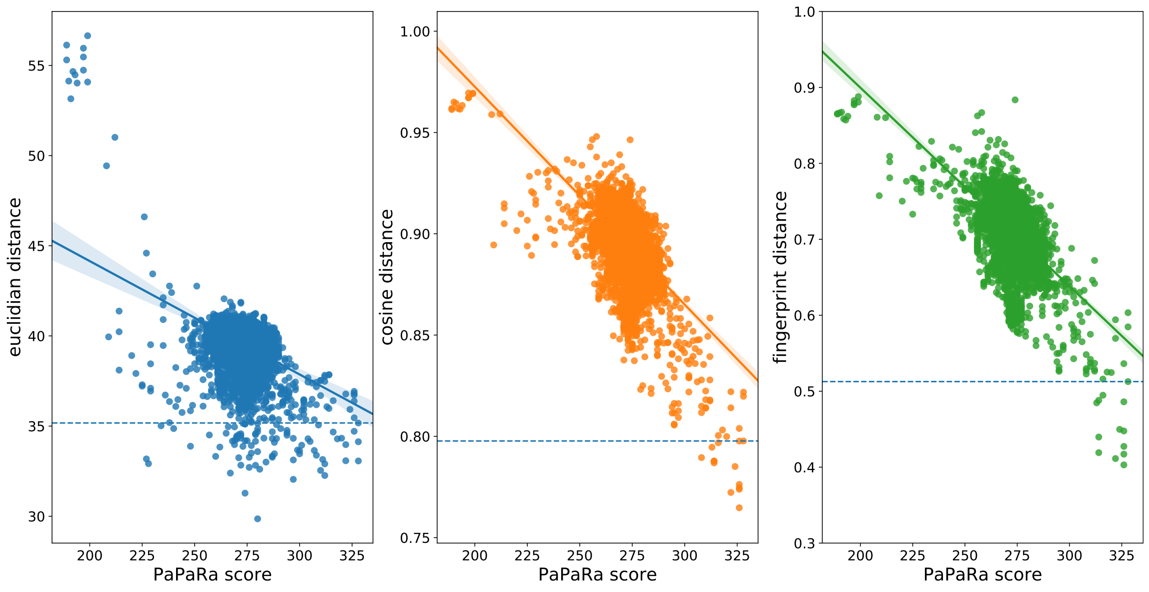

Figure 8.7: Comparison of the euclidean, cosine and fingerprint distance approximation plots. Approximation of the *PaPaRa* score by 7-mer distance functions on the tara data set for query 012.

The cosine distance classifies 95% of the queries with high success rate, but performs considerably worse for the highest threshold. In Figure 8.8, the greater effect of the worst performing queries is visible since the 98% success rate threshold moves the according distance to the right.
In contrast, the euclidean distance has a higher mean value, but if a more accurate classification is desired, it is superior to the cosine distance (the 98% threshold is further left in Figure 8.9).

This leads to the following trade-off: if faster computation is preferred over higher success rate, then the cosine distance should be chosen. Otherwise the euclidean distance is preferable.

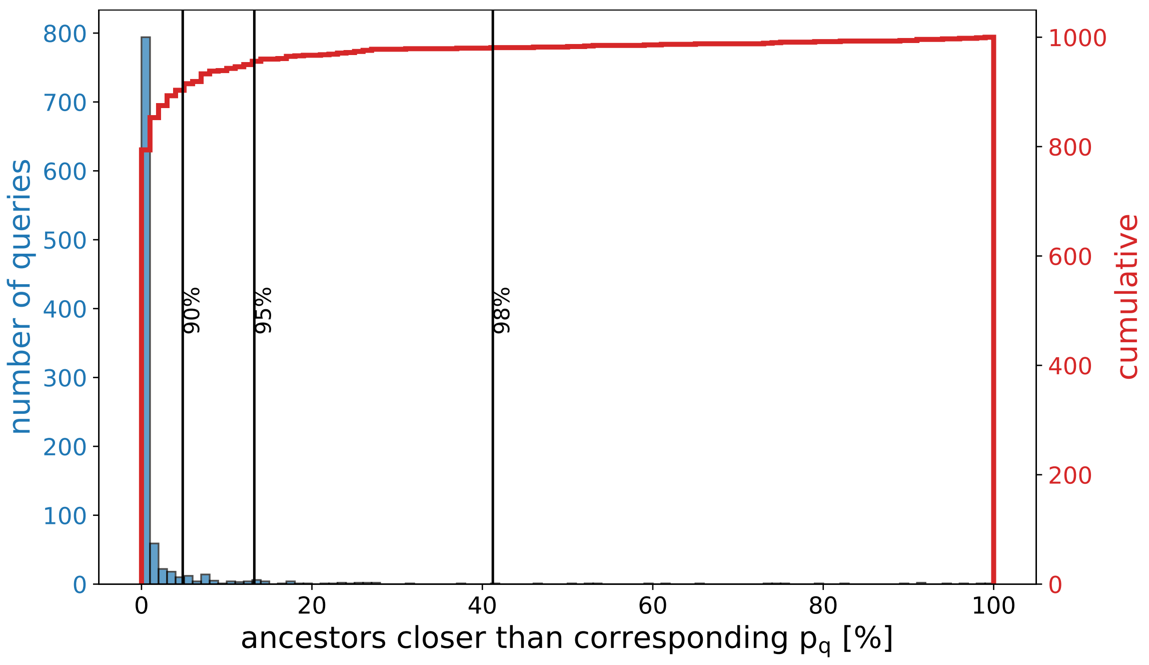

Figure 8.8: Efficiency of cosine 7-mer distance.
data set: tara
gaps removed from ancestors

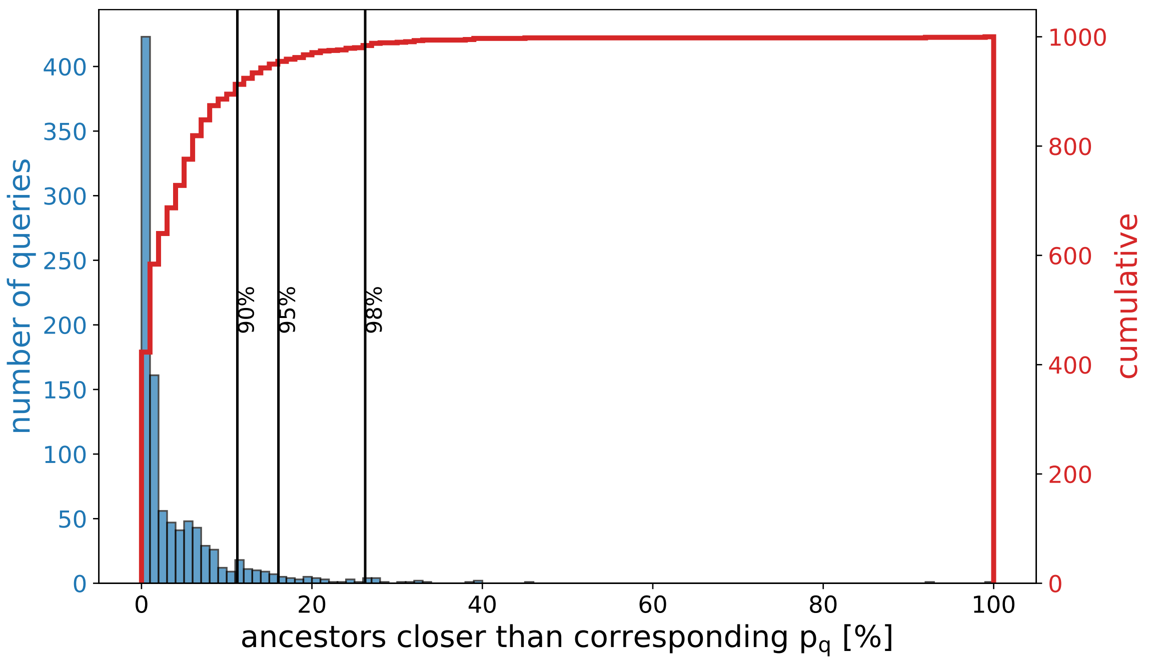

Figure 8.9: Efficiency of euclidean 7-mer distance.
data set: tara
gaps removed from ancestors

Subsection 8.2.1 Selection of k

Since the heuristics delivered more consistent results on the tara data set, it was also chosen for further tests on k-mer size.

With increasing k, the euclidean (Table 8.5) and the cosine distance (Table 8.6) improve regarding all properties, until a k-mer length of 8. From k-mer length 7 onwards, both yield diminishing returns. (As a reminder: the amount of possible k-mers is 4^k, therefore the vector size increases exponentially.)
In the case of the Jensen-Shannon divergence (Table 8.7), this effect already sets in at $k := 6$. Even a deterioration can be observed for $k \geq 8$.
The optimal k value for the fingerprint distance (Table 8.8) is 5 or 6, depending on the level of desired accuracy.

k-mer	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
3-mer	73.01	0.05	99.71	27.85	98.39	98.65	99.05
4-mer	58.44	0.07	99.76	31.87	97.96	98.51	99.03
5-mer	29.64	0.04	99.8	32.9	87.67	96.06	98.93
6-mer	7.53	0.01	99.79	14.15	18.23	35.07	58.25
7-mer	4.10	0.01	99.63	7.24	11.25	16.07	26.25
8-mer	3.61	0.01	53.41	5.04	9.70	14.13	18.50
9-mer	3.71	0.01	40.70	5.03	9.53	15.08	19.75

Table 8.5: Efficiency properties for the euclidean function. Distributions of $f_d(q)$ among all queries with euclidean distance and varying k on the tara data set.

k-mer	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
3-mer	25.41	0.01	99.48	28.82	77.49	87.09	93.90
4-mer	57.27	0.07	99.87	35.12	98.83	99.33	99.57
5-mer	6.11	0.01	98.43	15.21	16.87	38.90	69.61
6-mer	3.24	0.01	99.44	11.59	6.47	15.64	46.04
7-mer	2.79	0.01	99.01	11.21	4.84	13.24	41.23
8-mer	2.96	0.01	99.51	11.83	4.42	13.35	52.60
9-mer	3.25	0.01	99.79	12.74	5.38	14.84	56.79

Table 8.6: Efficiency properties for the cosine function. Distributions of $f_d(q)$ among all queries with cosine distance and varying k on the tara data set.

k-mer	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
3-mer	24.13	0.01	99.20	27.72	73.11	86.19	93.41
4-mer	14.86	0.01	98.91	22.68	50.11	74.50	86.05
5-mer	6.17	0.01	98.68	14.83	18.28	39.22	64.27
6-mer	4.24	0.01	99.43	13.85	7.55	26.66	68.88
7-mer	4.02	0.01	98.93	13.84	7.71	20.89	64.58
8-mer	4.44	0.01	99.01	14.85	7.65	25.10	70.80
9-mer	4.88	0.01	99.76	15.91	8.82	38.01	75.98

Table 8.7: Efficiency properties for the Jensen-Shannon function. Distributions of $f_d(q)$ among all queries with Jensen-Shannon divergence and varying k on the tara data set.

k-mer	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
3-mer	98.62	18.36	100.00	6.76	99.84	99.92	99.96
4-mer	22.15	0.05	95.18	20.00	51.05	64.37	76.30
5-mer	4.16	0.01	92.91	11.63	10.76	24.62	44.72
6-mer	4.17	0.01	99.48	13.82	7.71	25.58	68.46
7-mer	5.17	0.01	99.39	15.92	10.86	36.15	75.79
8-mer	5.87	0.01	99.87	17.49	15.43	43.13	81.18
9-mer	6.21	0.01	99.77	18.29	16.50	53.53	82.69

Table 8.8: Efficiency properties for the fingerprint function. Distributions of $f_d(q)$ among all queries with fingerprint distance and varying k on the tara data set.

Subsection 8.2.2 Alphabet Reduction

The idea from Section 5.1.2 turns out to be a dead end, since the performance of all heuristics decrease on sequences comprising a reduced alphabet for each data set. Only the results for the tara data set (see Table 8.9) are discussed in the following. The results for the other data sets are comparably bad.

Distances	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
Euclidean	14.55	0.03	95.45	19.84	54.98	60.88	67.13
Cosine	45.11	0.01	99.92	38.57	98.31	99.27	99.6
Jensen-Shannon	80.21	0.09	99.97	28.92	99.29	99.52	99.72
Fingerprint	97.1	44.07	99.99	6.32	99.75	99.85	99.93

Table 8.9: Efficiency properties per distance function. Distributions of $f_d(q)$ among all queries in the different 7-mer distance functions on the tara data set. Gaps were removed from the ancestors. The reduced alphabet was used.

The overall worse results with the reduced alphabet can be seen for the otherwise well performing euclidean (Figure 8.10) and cosine distance (Figure 8.11).

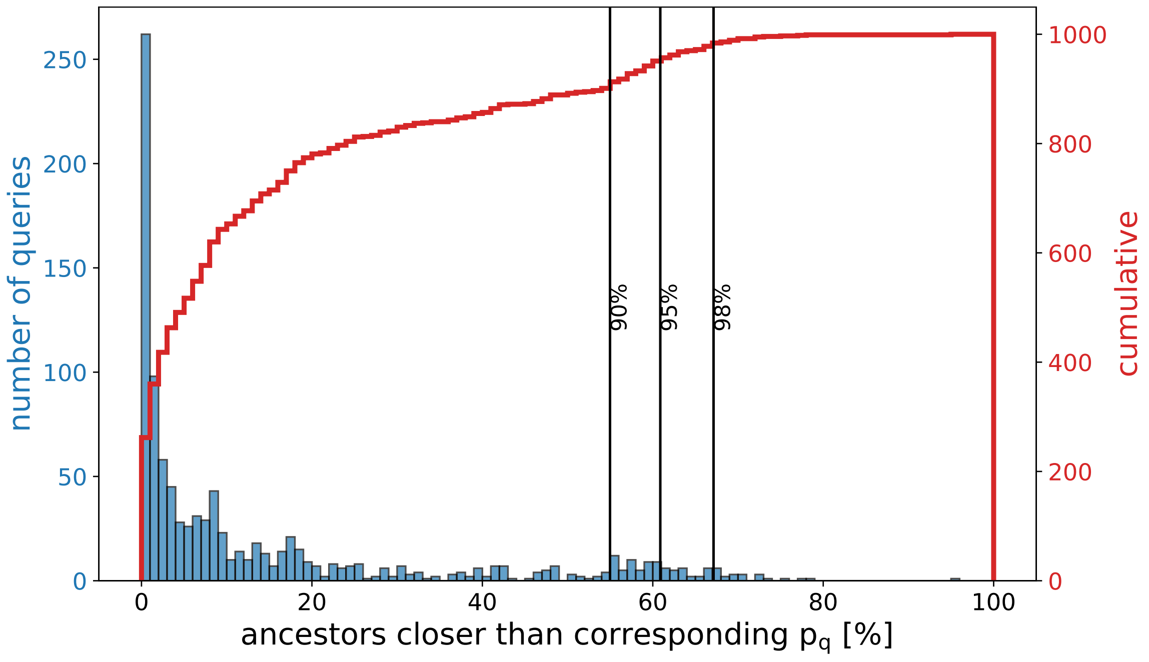

Figure 8.10: Efficiency of euclidean 7-mer distance.
data set: tara
gaps removed from ancestors
reduced alphabet

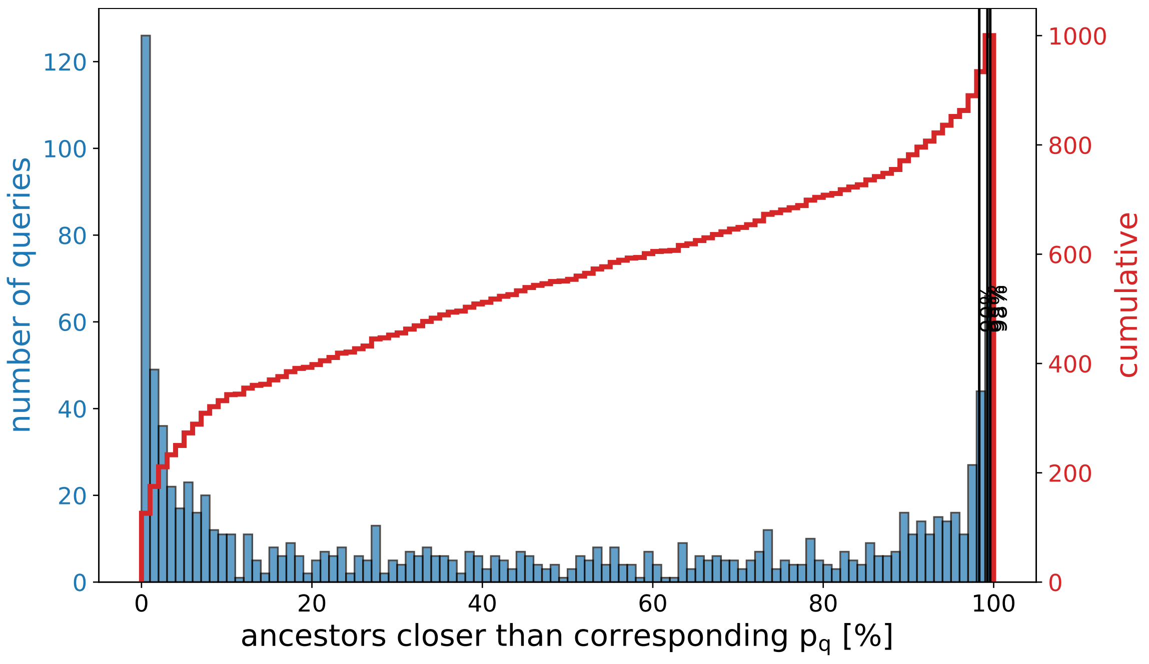

Figure 8.11: Efficiency of cosine 7-mer distance.
data set: tara
gaps removed from ancestors
reduced alphabet

Section 8.3 Bacterial Vaginosis Data Set

For the Bacterial Vaginosis data set, Table 8.10 shows remarkable performance by all heuristics except for the euclidean distance, even though the euclidean approximation shows analogous performance on other data sets. The histograms do not convey more information than Table 8.10, but are included in the appendix for the sake of completeness.

If the cosine distance is the heuristic of choice, then analyzing only 2% of the ancestors with *PaPaRa* suffices to find the optimal alignment for 98% of all queries. Even the worst query requires considering only 3.58% of the ancestors to find the optimal alignment.

Distances	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
Euclidean	6.84	0.06	28.79	7.09	18.86	22.44	23.19
Cosine	0.87	0.06	3.58	0.43	1.19	1.19	1.26
Jensen-Shannon	0.6	0.06	3.52	0.44	1.13	1.19	1.51
Fingerprint	1.02	0.06	3.27	0.46	1.45	1.45	1.45

Table 8.10: Efficiency properties per distance function. Distributions of $f_d(q)$ among all queries in the different 7-mer distance functions on the bv data set. Gaps were removed from the ancestors.

The fingerprint distance in Figure 8.12 shows that the distribution of query k-mers found in the ancestors, varies substantially for different ancestors. For low *PaPaRa* scores almost no query k-mer is found in the ancestor (left points are near 0.9 fingerprint distance) and high *PaPaRa* scoring ancestors contain almost all query k-mers (fingerprint distance is close to 0). In comparison, the fingerprint distance results in a much more narrow span of the ancestor distribution when applied to the neotrop data set (see Figure 8.6).

Interestingly, when varying the value for k for the cosine distance on this data set, near-optimal values are already achieved with $k := 3$ (see Table 8.11) as opposed to the tara data set. This is probably a result of the BV data set's properties.

k-mer	mean[%]	min[%]	max[%]	stddev[%]	90%[%]	95%[%]	98%[%]
2-mer	6.34	0.13	59.27	5.49	10.94	14.52	22.69
3-mer	2.20	0.06	26.46	2.38	3.21	4.71	7.98
4-mer	1.21	0.06	6.60	0.82	2.07	2.39	3.02
5-mer	0.99	0.06	3.52	0.52	1.38	1.51	1.63
6-mer	0.95	0.06	3.71	0.49	1.38	1.38	1.51
7-mer	0.87	0.06	3.58	0.43	1.19	1.19	1.26
8-mer	0.85	0.06	3.65	0.42	1.19	1.19	1.32

Table 8.11: Efficiency properties for the cosine function. Distributions of $f_d(q)$ among all queries with cosine distance and varying k on the BV data set.

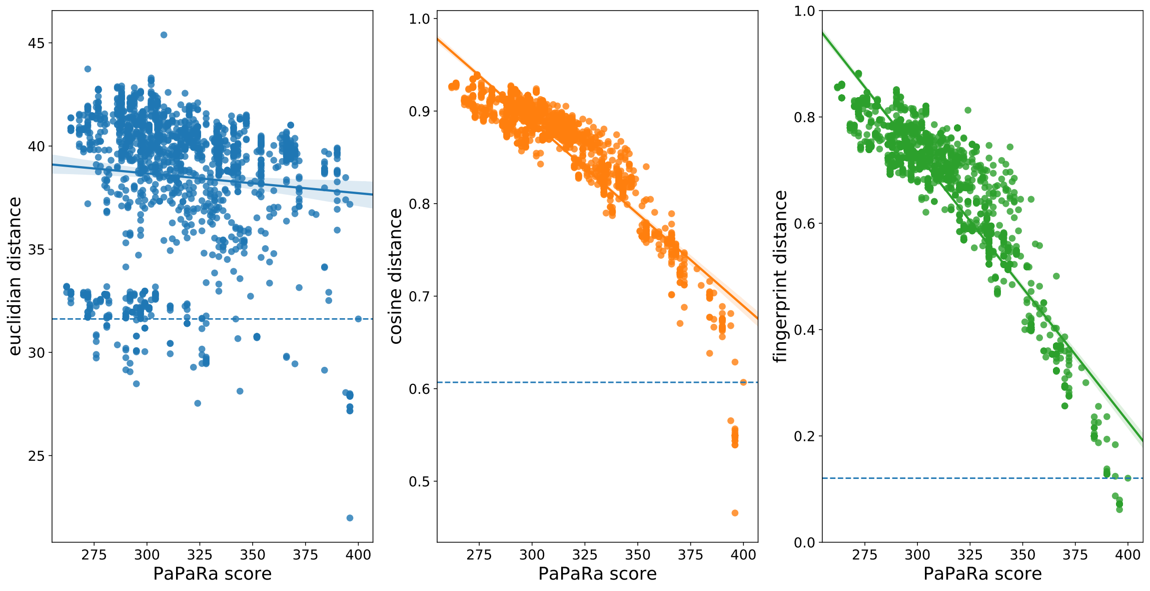

Figure 8.12: Comparison of the euclidean, cosine and fingerprint distance approximation plots. Approximation of the *PaPaRa* score by 7-mer distance functions on the bv data set for query 008.

Section 8.4 Summary

There are several takeaways from analyzing the heuristics on the different data sets:

- The best performing heuristic among the three data sets is the cosine distance, as observed in Table 8.2, Table 8.4 and Table 8.10.
- There is no "perfect" heuristic. One has to decide on how many queries are allowed to produce worse results than an unfiltered *PaPaRa* would have. Small adjustments to the threshold can result in large gains in computation time. This effect can be seen most prominently in the cosine distance (see, for example, Figure 8.3).
- The Jensen-Shannon divergence requires approximately twice as many computations as the other metrics. This additional cost is not justified by the results, as can, for example, be seen in Table 8.10.
- The intuitive idea to consider only mutual k-mers between the query and the ancestor and disregard the ancestor's excess k-mers, turns out to worsen the result for the neotrop and tara data sets. The fingerprint distance, which utilizes this concept, is mostly inferior to the euclidean distance (see for example Table 8.4).
- The most impactful parameter is removing characters from the ancestors that are marked by the gap flag vector. Comparing, for example, Table 8.1 and Table 8.2 leads to this conclusion. This is due to the fact that the heuristic tries to specifically imitate *PaPaRa*'s behaviour.
- In the case of the mostly superior cosine and euclidean heuristics, generally longer k-mers lead to better results. Since increasing k can lead to an exponential increase in computation time and yields *diminishing returns*, a value greater than 8 is not recommended.
- Reducing the alphabet, albeit useful in other applications, consistently leads to worse results in approximating *PaPaRa*.
- For the BV data set, it is conjectured that considering mutual k-mer matches between query and ancestor suffices to assess similarity. Unlike the other data sets, where there are fewer mutual k-mers (as can be seen by the fingerprint distance's behaviour), the distinct ancestor k-mers are of little consequence for *PaPaRa*. Having fewer and more distinct ancestors as in the BV data set, might explain the difference in behaviour among the data sets.
- To achieve a success rate of 95%, the following settings prove to work reliably among all three data sets:
  - metric: cosine
  - k-mer length: 3
  - remove gaps
  - rule out the 80% most distant ancestors

Chapter 9 Conclusion and Future Work

In this thesis, an improvement for the alignment tool *PaPaRa* was developed and analyzed. The discussed heuristics attempt to approximate *PaPaRa*'s evaluation, in order to reduce costly *PaPaRa* alignment calculations. Since the results vary substantially among data sets, a reliable prediction on the heuristics performance for other data sets cannot be made. Specific properties of the data sets could be the reason for those varying results. Formalizing these properties and finding correlations between the nature of the data and *PaPaRa*'s behaviour open up a line of further investigations. This could also be done on simulated data, where the properties are controllable. The recommended settings in this thesis could be used as a starting point.

The benchmark for the quality of the heuristics was *PaPaRa*. It remains to be investigated, if failing to find the highest *PaPaRa* scoring ancestor necessarily leads to a worse phylogenetic placement.

As with all k-mer based approaches, the algorithm does not take into account potential mutations which may have happened during evolution. This could be a weakness, that can not be improved through different distance measures. *PaPaRa*, however, takes this into account with the help of a costly calculation. In other words, there is a certain chance of ancestors receiving a good rating by *PaPaRa*, which is missed by k-mer based screening methods.

Furthermore, the fingerprint distance *FPD* value may be used to determine a promising value of k for the k-mer based approaches, and to estimate the mutation rate a query DNA sequence may have undergone. It may be useful to analyze other alignment-free methods that are not based on k-mer representations.

Certainly, the implementation has room for optimization. For example the resolution of the ambiguous k-mers sometimes produces densely filled vectors, so using the C++ unordered map as a sparse vector representation may yield sub-optimal runtimes.

Bibliography

[AHK01] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. “On the Surprising Behavior of Distance Metrics in High Dimensional Space”. In: *International conference on database theory.* Springer. 2001, pp. 420–434.

[Alm+01] Jonas S. Almeida et al. “Analysis of Genomic Sequences by Chaos Game Representation”. In: *Bioinformatics* 17 5 (2001), pp. 429–37.

[Alt+90] Stephen F Altschul et al. “Basic Local Alignment Search Tool”. In: *Journal of molecular biology* 215.3 (1990), pp. 403–410.

[Bar+18] Pierre Barbera et al. “EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences”. In: *Systematic Biology* 68.2 (Sept. 2018), pp. 365–369.

[Ber16] Simon Berger. *papara_nt, New Codebase of the PaPaRa Algorithm.* `https://github.com/sim82/papara_nt`. 2016.

[BS11] Simon A Berger and Alexandros Stamatakis. “Aligning Short Reads to Reference Alignments and Trees”. In: *Bioinformatics* 27.15 (2011), pp. 2068–2075.

[BS12] Simon A Berger and Alexandros Stamatakis. “PaPaRa 2.0: A Vectorized Algorithm for Probabilistic Phylogeny-Aware Alignment Extension”. In: *Heidelberg Institute for Theoretical Studies* (2012).

[BSB13] Oliver Bonham-Carter, Joe Steele, and Dhundy Bastola. “Alignment-free Genetic Sequence Comparisons: A Review of Recent Approaches by Word Analysis”. In: *Briefings in bioinformatics* 15.6 (2013), pp. 890–905.

[CPM18] D.P. Clark, N.J. Pazdernik, and M.R. McGehee. *Molecular Biology.* Elsevier Science, 2018. ISBN: 9780128132883. URL: `https://books.google.de/books?id=Fd9MtAEACAAJ`.

[Dai16] Jeff Daily. “Parasail: SIMD C Library for Global, Semi-global, and Local Pairwise Sequence Alignments”. In: *BMC bioinformatics* 17.1 (2016), p. 81.

[Fel81] Joseph Felsenstein. “Evolutionary Trees from DNA Sequences: A Maximum Likelihood Approach”. In: *Journal of molecular evolution* 17.6 (1981), pp. 368–376.

[Fit71] Walter M Fitch. “Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology”. In: *Systematic Biology* 20.4 (1971), pp. 406–416.

[Got82] Osamu Gotoh. “An Improved Algorithm for Matching Biological Sequences”. In: *Journal of molecular biology* 162.3 (1982), pp. 705–708.

[HJ12] Roger A Horn and Charles R Johnson. *Matrix Analysis*. Cambridge university press, 2012.

[HR07] Michael Höhl and Mark A Ragan. “Is Multiple-Sequence Alignment Required for Accurate Inference of Phylogeny?” In: *Systematic Biology* 56.2 (2007), pp. 206–221.

[Hun07] J. D. Hunter. “Matplotlib: A 2D Graphics Environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: `10.1109/MCSE.2007.55`.

[Jef90] H Joel Jeffrey. “Chaos Game Representation of Gene Structure”. In: *Nucleic acids research* 18.8 (1990), pp. 2163–2170.

[Jün+13] Sebastian Jünemann et al. “Updating Benchtop Sequencing Performance Comparison”. In: *Nature biotechnology* 31.4 (2013), p. 294.

[KG01] Liisa B. Koski and G. Brian Golding. “The Closest BLAST Hit Is Often Not the Nearest Neighbor”. In: *Journal of Molecular Evolution* 52.6 (June 2001), pp. 540–542.

[KL51] Solomon Kullback and Richard A Leibler. “On Information and Sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[Klu+16] Thomas Kluyver et al. “Jupyter Notebooks- A Publishing Format for Reproducible Computational Workflows.” In: *ELPUB*. 2016, pp. 87–90.

[Kne+19] C. W. Knetsch et al. “DNA Sequencing”. In: *Molecular Diagnostics: Part 1: Technical Backgrounds and Quality Aspects*. Ed. by E. van Pelt-Verkuil, W.B. van Leeuwen, and R. te Witt. Singapore: Springer Singapore, 2019, pp. 339–360. ISBN: 978-981-13-1604-3.

[Lin91] Jianhua Lin. “Divergence Measures Based on the Shannon Entropy”. In: *IEEE Transactions on Information theory* 37.1 (1991), pp. 145–151.

[LS12] Ben Langmead and Steven L Salzberg. “Fast Gapped-read Alignment with Bowtie 2”. In: *Nature methods* 9.4 (2012), p. 357.

[Mah+17] Frédéric Mahé et al. “Parasites Dominate Hyperdiverse Soil Protist Communities in Neotropical Rainforests”. In: *Nature Ecology & Evolution* 1.4 (2017), p. 0091.

[McK10] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51–56.

[ME13] Frederick A Matsen IV and Steven N Evans. “Edge Principal Components and Squash Clustering: Using the Special Structure of Phylogenetic Placement Data for Sample Comparison”. In: *PloS one* 8.3 (2013), e56859.

[MKA10] Frederick A Matsen, Robin B Kodner, and E Virginia Armbrust. “pplacer: Linear Time Maximum-Likelihood and Bayesian Phylogenetic Placement of Sequences onto a Fixed Reference Tree”. In: *BMC bioinformatics* 11.1 (2010), p. 538.

[NW70] Saul B Needleman and Christian D Wunsch. “A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins”. In: *Journal of molecular biology* 48.3 (1970), pp. 443–453.

[Par+15] Daniel J Park et al. "Ebola Virus Epidemiology, Transmission, and Evolution during Seven Months in Sierra Leone". In: *Cell* 161.7 (2015), pp. 1516–1526.

[Rog11] Torbjørn Rognes. "Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation". In: *BMC bioinformatics* 12.1 (2011), p. 221.

[Roz] Gennadiy Rozental. *Boost Test Library: The Unit Test Framework.* `https://www.boost.org/doc/libs/1_45_0/libs/test/doc/html/utf.html`. Accessed: 2019-10-25.

[Ruc+18] Enzo Rucci et al. "SWIFOLD: Smith-Waterman Implementation on FPGA with OpenCL for Long DNA Sequences". In: *BMC systems biology* 12.5 (2018), p. 96.

[Sim+09] Gregory E Sims et al. "Alignment-free Genome Comparison with Feature Frequency Profiles (FFP) and Optimal Resolutions". In: *Proceedings of the National Academy of Sciences* 106.8 (2009), pp. 2677–2682.

[SJ08] Jay Shendure and Hanlee Ji. "Next-Generation DNA Sequencing". In: *Nature biotechnology* 26.10 (2008), p. 1135.

[SNC77] Frederick Sanger, Steven Nicklen, and Alan R Coulson. "DNA Sequencing with Chain-Terminating Inhibitors". In: *Proceedings of the national academy of sciences* 74.12 (1977), pp. 5463–5467.

[Sri+12] Sujatha Srinivasan et al. "Bacterial Communities in Women with Bacterial Vaginosis: High Resolution Phylogenetic Analyses Reveal Relationships of Microbiota to Clinical Criteria". In: *PloS one* 7.6 (2012), e37818.

[Ste09] William J Stewart. *Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling.* Princeton university press, 2009, pp. 257–259.

[Sun+15] Shinichi Sunagawa et al. "Structure and Function of the Global Ocean Microbiome". In: *Science* 348.6237 (2015), p. 1261359.

[Was+18] Michael Waskom et al. *mwaskom/seaborn: v0.9.0 (July 2018).* Version v0.9.0. Accessed: 2019-10-25. July 2018. URL: `https://doi.org/10.5281/zenodo.1313201`.

[Wet] KA Wetterstrand. *DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP).* `www.genome.gov/sequencingcostsdata`. Accessed: 2019-10-25.

[Wik19a] Wikibooks. *C++ Programming/Code/Design Patterns — Wikibooks, The Free Textbook Project.* `https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/Design_Patterns&oldid=3520978`. Accessed: 2019-10-25. 2019.

[Wik19b] Wikipedia contributors. *List of sequence alignment software — Wikipedia, The Free Encyclopedia.* `https://en.wikipedia.org/w/index.php?title=List_of_sequence_alignment_software&oldid=922841378`. Accessed: 2019-10-25. 2019.

[Zie+17] Andrzej Zielezinski et al. "Alignment-free Sequence Comparison: Benefits, Applications, and Tools". In: *Genome biology* 18.1 (2017), p. 186.

Appendix

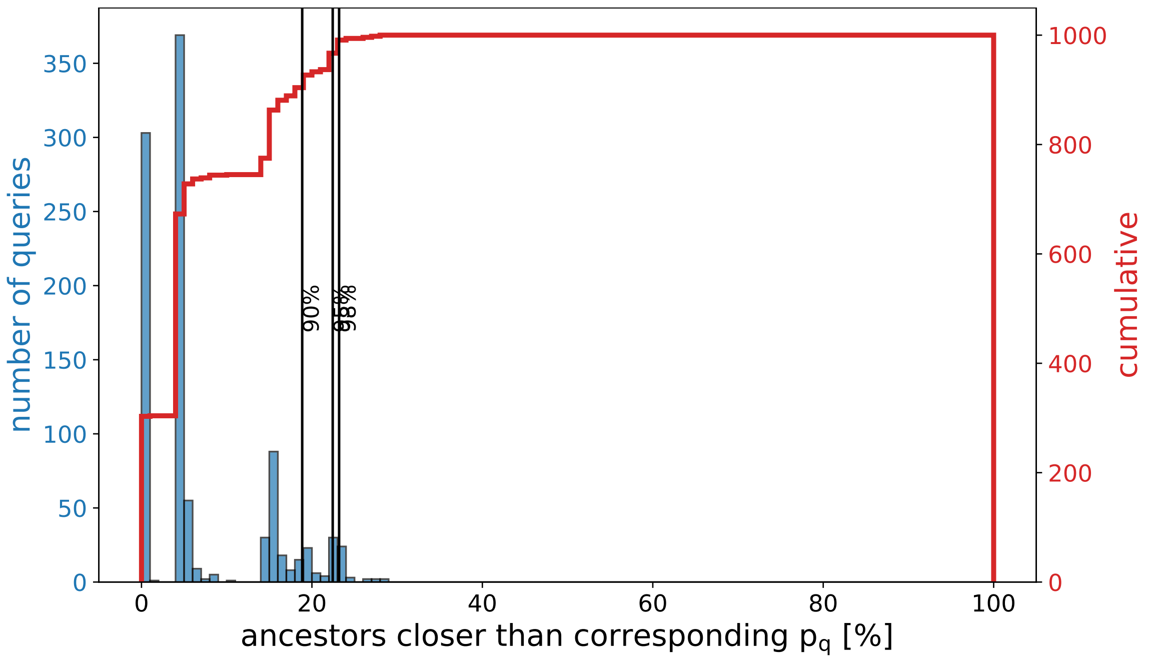

Figure .1: Efficiency of euclidean 7-mer distance.
data set: bv
gaps removed from ancestors

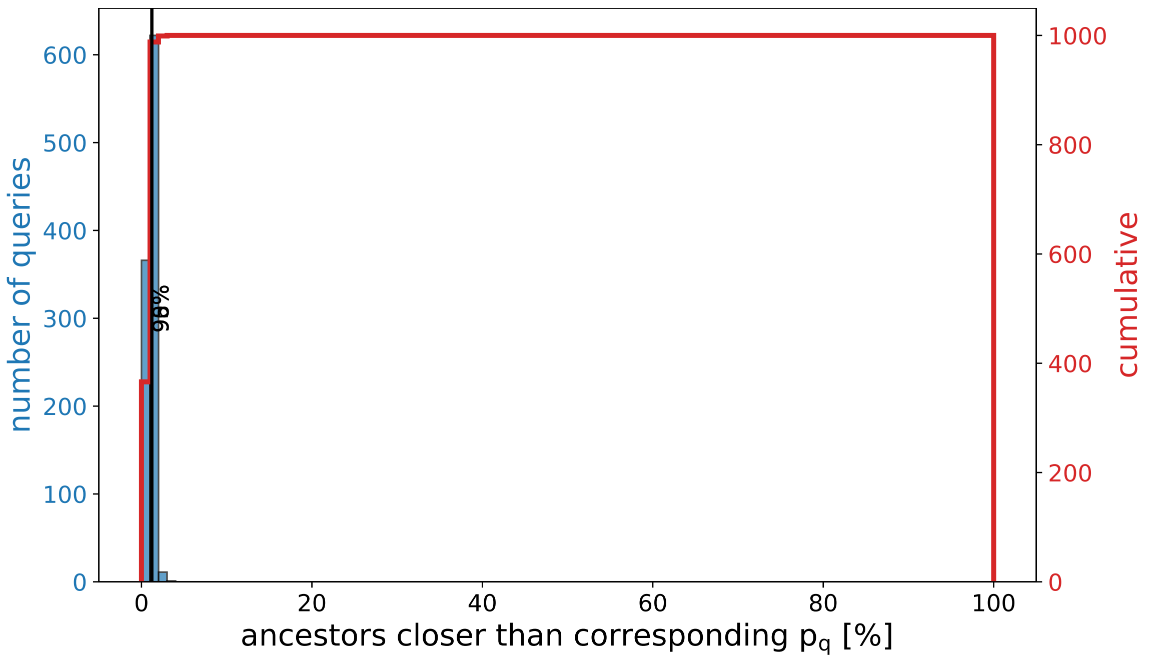

Figure .2: Efficiency of cosine 7-mer distance.
data set: bv
gaps removed from ancestors