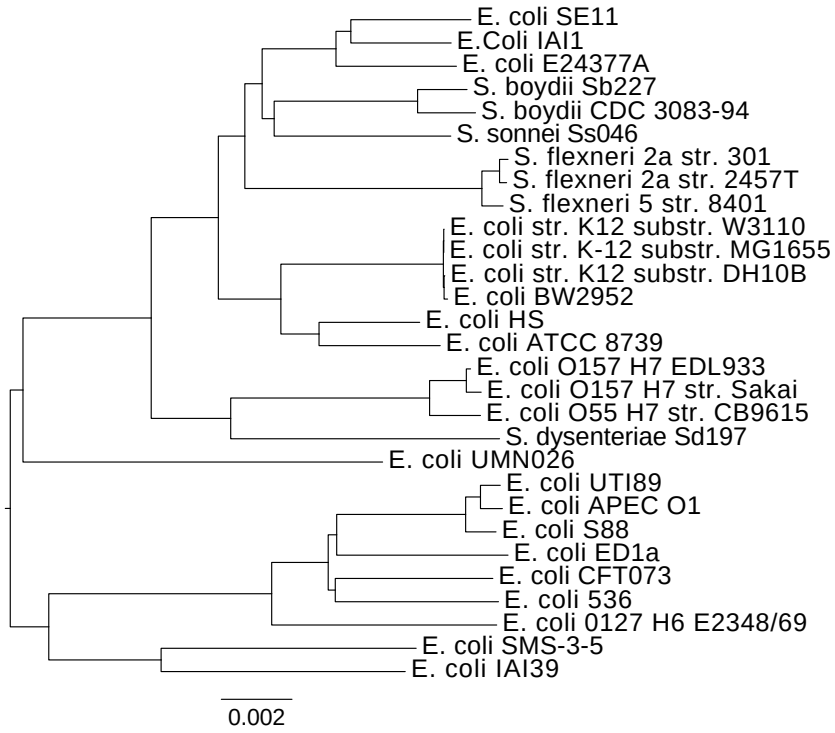
ANDI: the anchor distance

Efficient Alignment-Free Estimation of Evolutionary Distances

https://github.com/EvolBioInf/andi

Fabian Klötzl
kloetzl@evolbio.mpg.de

Version 0.9, 2015-04-28



Abstract

This is the documentation of the ANDI program for estimating the evolutionary distance between closely related genomes. It efficiently and accurately computes a distance for substitution rates up to 0.5. These distances are based on ungapped local alignments framed by anchors. Anchors are efficiently found using an enhanced suffix array. As a result, ANDI scales well, even for data sets containing thousands of bacterial genomes.

License

Contents

1 Installation

The easiest way to obtain your own copy of ANDI, is to download the latest release from GitHub.[1] Please note, that ANDI requires LIBDIVSUFSORT[2] for optimal performance. If you wish to install ANDI without LIBDIVSUFSORT, consult Section **??**.

1.1 Installing from Source Package

Once you have downloaded the package, unzip it and change into the newly created directory.

```
~ % tar -xzvf andi-0.9.tar.gz
~ % cd andi-0.9
```

You should now see plenty of files, of which most better stay untouched. If you feel that this documentation is too verbose, and you prefer more succinct instructions, please consult the README. However, if you are still reading this, execute the following commands, to proceed.

```
~/andi-0.9 % ./configure
~/andi-0.9 % make
~/andi-0.9 % sudo make install
```

This installs ANDI for all users on your system. If you do not have root privileges, you will find a working copy of ANDI in the `src` subdirectory. For the rest of this documentation, I will assume, that ANDI is in your $PATH.

Now ANDI should be ready for use. Try invoking the help.

```
~/andi-0.9 % andi -h
Usage: andi [-jrv] [-p FLOAT] FILES...
     FILES... can be any sequence of FASTA files. If no files are
         supplied, stdin is used instead.
Options:
 -j, --join   Treat all sequences from one file as a single genome
 -m, --low-memory Use less memory at the cost of speed
 -p <FLOAT>   Significance of an anchor pair; default: 0.05
 -r, --raw    Calculates raw distances; default: Jukes-Cantor
    corrected
 -v, --verbose Prints additional information
 -t <INT>     The number of threads to be used; default: 1
 -h, --help   Display this help and exit
    --version Output version information and acknowledgments
```

ANDI also comes with a man page, which can be accessed via **`man andi`**. But once you are done with this documentation, you will require it scarcely.

1.2 Installing without LIBDIVSUFSORT

If you cannot or do not want to install LIBDIVSUFSORT, ANDI has got you covered. It comes with its own implementation for a *suffix array construction algorithm*. That one is called PSUFSORT and features its own GIT repo.[3] To activate it for ANDI, proceed as described in Section **??**, but with the following twist:

[1] https://github.com/EvolBioInf/andi/releases
[2] https://github.com/y-256/libdivsufsort
[3] https://github.com/kloetzl/psufsort

```
~/andi % ./configure --without-libdivsufsort
```

Please note that this requires a C++11 compiler. Also, PSUFSORT may be slower than LIBDIVSUFSORT and thus lead to inferior runtimes.

1.3 Installing from Git Repository

To build ANDI from the GIT repo, execute the following steps.

```
~ % git clone git@github.com:EvolBioInf/andi.git
~ % cd andi
~/andi % autoreconf -i
```

For old versions of AUTOCONF you may instead have to use `autoreconf -i -Im4`.
Continue with the GNU trinity as in Section **??**.

2 Usage

The input sequences for ANDI should be in FASTA format. Any number of files can be passed. Each file can contain more than one sequence.

```
~ % andi S1.fasta S2.fasta
2
S1      0.0000 0.0979
S2      0.0979 0.0000
```

If no file argument is given, ANDI reads the input from STDIN. This makes it convenient to use in UNIX pipelines.

```
~ % cat S1.fasta S2.fasta | andi
2
S1      0.0000 0.0979
S2      0.0979 0.0000
```

The output of ANDI is a matrix in PHYLIP style: On the first line, the number of compared sequences is given. Then the matrix is printed, where each line is proceeded by the name of the ith sequence. Note that the matrix is symmetric and the main diagonal contains only zeros. The numbers themselves are evolutionary distances, estimates as substitution rates.

2.1 Input

As mentioned before, ANDI reads in FASTA files. It recognizes only the four standard bases, case insensitive (`[acgtACGT]`). All other nucleotides are excluded from the analysis and ANDI prints a warning, when this happens.

Today, a lot of genome projects to not end with complete genomes; instead short contigs for the genomes are given. If all contigs of a genome are in a single FASTA file, ANDI will treat them as belonging together, when switched to JOIN mode. This can be achieved by using the `-j` or `--join` command line switch.

```
~ % andi --join E_coli.fasta Shigella.fasta
[Output]
```

When the JOIN mode is active, the file names are used to name the individual sequences. Thus, in JOIN mode, each genomes has to be in its own file, and furthermore, at least one filename has to be given via the command line.

If ANDI seems to take unusually long, or requires huge amounts of memory, then you might have forgotten the JOIN switch. This makes ANDI compare each contig instead of each genome, resulting in much more comparisons! To make ANDI output the number of genome it wants to compare use the `--verbose` switch.

2.2 Output

The output of ANDI is written to `stdout`. This makes it easy to use on the command line and within shell scripts. As seen before, the matrix, computed by ANDI, is given in PHYLIP format.

```
~ % cat S1.fasta S2.fasta | andi
2
S1      0.0000 0.0979
S2      0.0979 0.0000
```

If the computation completed successful, ANDI exits with the status code 0. Otherwise, the `errno` is used as the exit code. ANDI can also produce a lot of warnings and error messages for the users convenience. These messages are printed to `stderr` and thus do not interfere with the normal output. Please consult your shells manual for redirection of streams.

2.3 Options

ANDI takes a small number of commandline options, of which even fewer are of interest on a day-to-day basis. If **`andi`** `-h` displays a `-t` option, then ANDI was compiled with multi-threading support (implemented using OpenMP). By default ANDI uses all available processors. However, if you want to restrict the number of threads use `-t`.

```
~ % time andi ../test/1M.1.fasta -t 1
2
S1      0.0000 0.0995
S2      0.0995 0.0000
./andi ../test/1M.1.fasta 0,60s user 0,01s system 99% cpu 0,613 total
~ % time andi ../test/1M.1.fasta -t 2
2
S1      0.0000 0.0995
S2      0.0995 0.0000
./andi ../test/1M.1.fasta -t 2 0,67s user 0,03s system 195% cpu 0,362
    total
```

Note that in the above examples the runtime reduced from 0.613 s, to 0.362 s using two threads. Making ANDI use more threads than input genomes leads to no further speed improvement.

The other most-important option is `--join`. It is used, when the genomes consist of multiple contigs. For more information, see section **??**.

By default, the distances computed by ANDI are *Jukes-Cantor* corrected. This means, the output is substitution rates, which is slightly larger than the mismatch rate. If you want to calculate the pure mismatch rate, ANDI can do so if told so by the `--raw` switch.

2.4 Example: ECO29

Here is a simple real-world example on how to use ANDI. As a data set we'll be using ECO29; 29 genomes of *E. Coli* and *Shigella*. You can download the data here: `http://guanine.evolbio.mpg.de/andi/eco29.fasta.gz`. The genomes have an average length of 4.9 million nucleotides amounting to a total 138 MB.

ECO29 comes a single FASTA file, where each sequence is a genome. Applying ANDI to this data set is straightforward.

```
~ % andi eco29.fasta > eco29.mat
andi: The input sequences contained characters other than acgtACGT.
   These were automatically stripped to ensure correct results.
```

The ECO29 data set includes non-canonical nucleotides, such as Y, N, and P. Unfortunately, ANDI cannot process these, thus they get stripped from the input sequences. The resulting matrix is stored in `eco29.mat`. Because the complete matrix would be too big for this page, here is a small excerpt.

```
~ % head -n 5 eco29.mat | cut -d ' ' -f 1-5
29
gi|563845 0.0000e+00 1.8388e-02 1.8439e-02 2.6398e-02
gi|342360 1.8388e-02 0.0000e+00 4.4029e-04 2.6166e-02
gi|300439 1.8439e-02 4.4029e-04 0.0000e+00 2.6123e-02
gi|261117 2.6398e-02 2.6166e-02 2.6123e-02 0.0000e+00
```

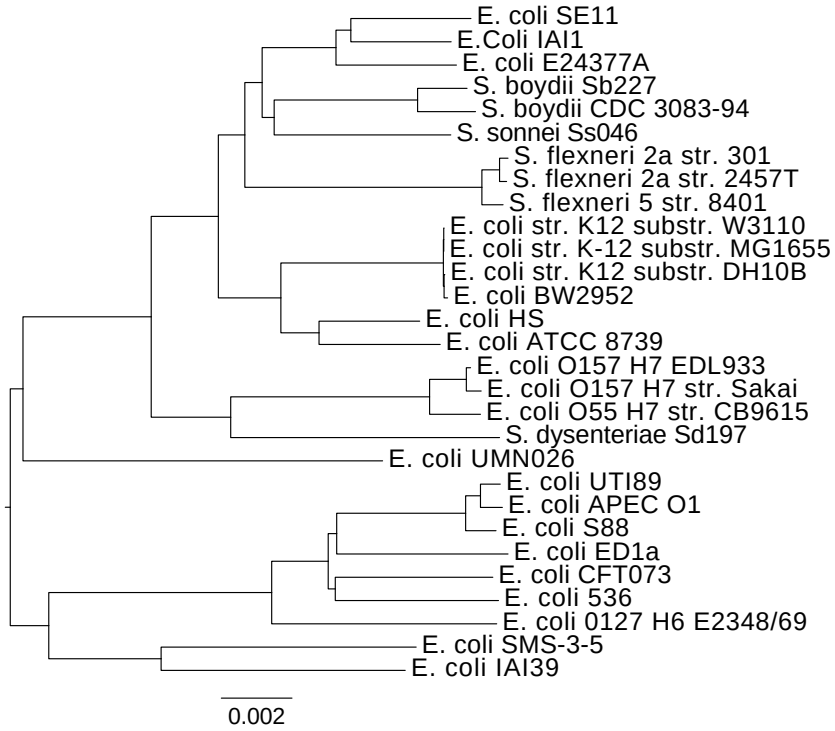
From this we now have to compute a tree via neighbor-joining. For this, I use a wrapper of PHYLIP called EMBASSY.[1] Because I enjoy rooted phylograms, I also midpoint-root the tree.

```
~ % fneighbor -datafile eco29.mat -outfile eco29.phylipdump
~ % fretree -spp 29 -intreefile eco29.treefile -outtreefile eco29.
    tree <<EOF
M
X
Y
R
EOF
```

The file `eco29.tree` now contains the tree in Newick format. Finally, the resulting phylogeny can be plotted using any tool of your choice.

```
~ % figtree eco29.tree &
```

Here is the tree I computed. Yours should look similar.



[1] http://emboss.sourceforge.net/embassy/#PHYLIP

3 The Anchor Distance in Detail

This section explains the idea of the *anchor distance*. After doing this in detail, we also compare this process with other *alignment-free* tools and verify the results with reference alignments. Parts of this section come from my master thesis which I will provide at request.

3.1 Motivation

Computing a phylogeny is a standard task arising in the analysis of genomes. There are various approaches to compute such a phylogeny. Often, this is done by computing an (multiple) alignment and then creating a matrix of evolutionary distances.
Interestingly, when the evolutionary distances are calculated from an alignment, indels are usually ignored. This means, the time which was spent on computing gaps is a waste of effort. If instead one can estimate the substitution rate directly from the data, the resulting pipeline will be much faster. This is essentially the idea of most *alignment-free* distance estimators.

3.2 Definition

The anchor distance is computed from two sequences, one called the *subject* S and a *query* Q with $Q, S \in \{A, C, G, T\}^*$[1]. Due to evolutionary events like *gene duplication*, the comparison with our yet-to-be-defined anchor distance d_{asym} may not be symmetric (i. e., $d_{asym}(Q, S) \neq d_{asym}(S, Q)$). To overcome this limitation, the final distance is the average of both comparisons.

Definition 1 (Anchor Distance)**.** *Let S_1 and S_2 be two genetic sequences. Then the* anchor distance *is the average of the two asymmetric comparisons with d_{asym}.*

$$d_a(S_1, S_2) = \frac{d_{asym}(S_1, S_2) + d_{asym}(S_2, S_1)}{2}$$

To compute the asymmetric anchor distance of S and Q, generate the **ESA!** (**ESA!**) of the subject, concatenated with its *reverse complement*. Then Q is streamed against S as follows. Set q to 0 and continue until it runs past $|Q|$. Compute the longest match of $Q[q..]$ with S. Continue finding matches and each time incrementing q by the length of the match until an anchor is found.

Definition 2 (Anchor)**.** *Let $Q[q..]$ be a match which cannot be further extended to the right. If it is unique in S and of some minimal length L, it is termed an* anchor.

Save the characteristics of the first anchor and keep finding matches until a second anchor is found. Unless the anchors form an *anchor pair*, replace the saved state of the first anchor with the second and try finding another second anchor. The anchors form a pair if they are equidistant, that is, their distance on Q is the same as for their counterparts on S (see Figure **??**).

An anchor pair frames a region of nucleotide sequence, which is assumed to be homologous. Since the two sequence parts are of equal length, a Hamming distance can be easily computed. More precisely, the number of homologous nucleotides and **SNP!**s are counted. These numbers are cumulated for every additional anchor pair found. The final anchor distance is the Jukes-Cantor corrected Hamming distance.

[1] Even though the genomic alphabet contains only the four characters A, C, G and T, the actual alphabet used by ANDI has the following additional characters $\{!, ;, \#, \backslash 0\}$. The $\#$ is used to separate a genome sequence from its reverse complement. Both of which might be made up of multiple contigs separated by ! and ;, respectively.

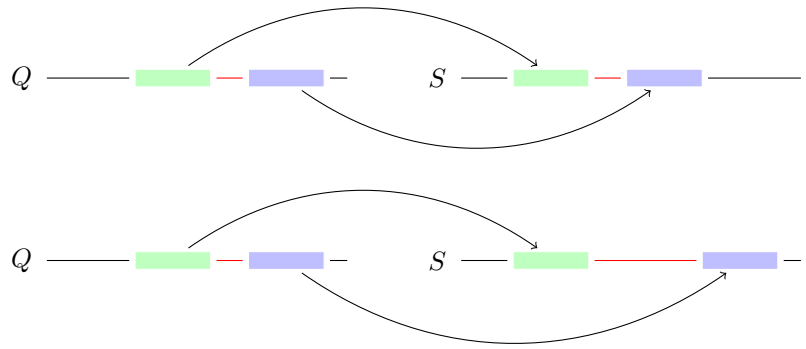


Figure 3.1: Anchor Pairs. In the upper panel the anchors on Q and S are equally spaced and hence considered a valid anchor pair. Thus, the **SNP!**s in the framed segment, shown in red, are counted. For the second figure the anchors are not equidistant and therefore ignored.

Definition 3 (Corrected Asymmetric Anchor Distance)**.**

$$d_{asym}(S,Q) = JC\left(\frac{\#SNP}{\#HomologousNucl}\right)$$

Listing **??** shows the pseudocode algorithm to compute d_{asym} using an **ESA!**. Various exceptions may arise during this calculation, which need to be handled, individually.

- The query and the sequence might be identical or the former might be contained in the latter. This leads to a single match extending over the full query; $d_{asym}(S,Q) = 0$.
- With very closely related sequences only a single anchor pair might extend over the query completely; $d_{asym}(S,Q) = JC(d_H(S,Q))$.
- If the query contains a subsequence multiple times that is only found once in the subject (e. g., gene duplication), the same part of the subject might be accounted for a homologous sequence more than once. Eventually, the count for homologous nucleotides might exceed the length of the subject. In this case, the distance is set to the special error value NaN.
- With very diverse, or totally unrelated sequences, no anchor pair may be found. In these cases, the distance is also set to NaN (see Section **??**).
- If an anchor could serve as both, a left and right anchor, be sure to count its nucleotides only once, to avoid biasing the result.

In addition to the anchor distance, ANDI computes another characteristic, the *coverage*, that is, the relative amount of homologous nucleotides. This is useful for debugging, but not accurate enough to serve as a distance in its own right.

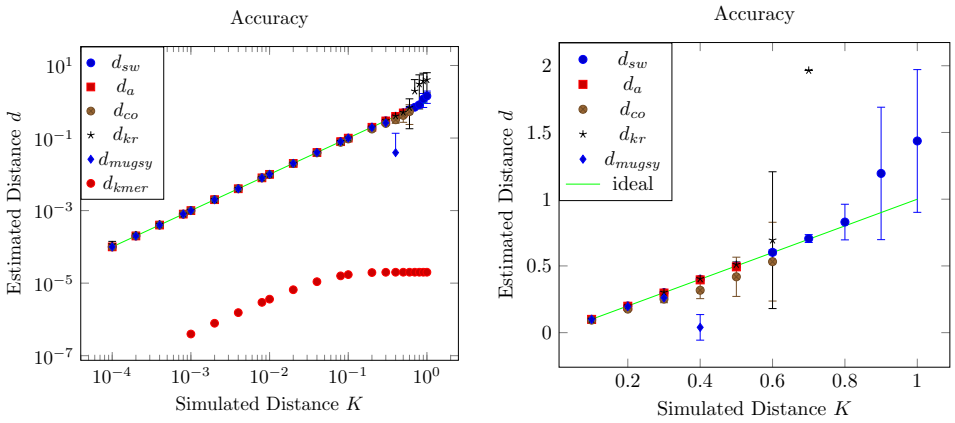
3.3 Accuracy

Figure **??** shows measurements for the previously described distances. Each data point is the mean of one hundred runs. For each run a sequence pair of length 100 kbp is simulated with a substitution rate K (Jukes-Cantor corrected). An ideal distance estimation method would calculate the exact substitution rate of the input and hence, have all its data points on the straight line.

As can be seen in Subfigure **??**, the k-mer based estimation is monotone, but at least one order of magnitude smaller than expected. This lack of accuracy makes it inferior to all other methods.

```
fn dist_anchor
requires S
input Q

let E ← ESA(S)
let L ← threshold(S,Q)
let Snps ← 0
let Homol ← 0

let last_pos_q ← 0
let last_match ← ⊥
let last_was_right_anchor ← false

let q ← 0
while q < |Q| do // Stream the complete query

  // Find the next match
  m ← get_match(E, Q[q...])
  if m.isUnique and m.length ≥ L then

    // m is an anchor
    if q - last_pos_q = m.pos - last_match.pos then
      // We have found a pair
      Snps ← Snps + count_diff(Q[last_pos_q...q], S[last_match.pos...m.
          pos])
      Homol ← Homol + q - last_pos_q
      last_match ← m
      last_was_right_anchor ← true
    else
      // Correctly count the nucleotides from right anchors
      if last_was_right_anchor = true then
        Homol ← Homol + last_match.length
      end

      last_was_right_anchor ← false
    end

    // Cache values for later
    last_pos_q ← q
    last_match ← m
  end

  // Skip the mutation
  q ← q + m.length + 1
end

output Snps/Homol
```

Listing 3.1: This algorithm computes the uncorrected asymmetric anchor distance of Q with respect to the subject S.



(a) log/log scale for 0.0001 to 1

(b) linear scale for 0.1 to 1

Figure 3.2: Estimation of the substitution rates for different distances. Shown are means and variance. An ideal estimator would have all its data points on the straight line. For the used implementations and parameters consult Table **??**.

The high number of substitutions makes good estimations for all methods increasingly difficult beyond a rate of 0.4. For a higher K value, d_{co} becomes downwards biased and stops working at 0.7. d_a fails beyond rates of $K \geq 0.5$. d_{kr} rapidly overestimates the distance for $K \geq 0.7$. For improved clarity its datapoints beyond $K = 0.7$ are omitted in Subfigure **??**. The best results for high substitution rates are produced by d_{sw}. Its estimations are reliable up to $K = 0.8$. Beyond that, they become upwards biased and start fluctuating heavily.

As a reference, MUGSY was used to compute alignments under the same conditions [**?**]. From the alignment, the program DNADIST from the PHYLIP toolbox was used to compute Jukes-Cantor corrected distances [**?**]. Unsurprisingly, the alignment is among the most accurate estimations up to $K = 0.3$. For $K = 0.4$ its reported distances are one order of magnitude too small. For bigger K, no alignment is produced.

3.4 Insertions and Deletions

We have already discussed in Section **??**, that ANDI may be sensitive to indels. To further explore for this issue, we simulated pairs of sequences with a fixed distance and varying indel rate. Figure **??** displays the results of this test.

For each data point in Figure **??**, two sequences of length 100 000 bp were simulated with a substitution rate $\pi = 0.1$ and varying indel rate ϕ. Now there are two equally correct measures of the *evolutionary distance*; the substitution rate π and the total error rate $\pi + \phi$.

The substitution rate has been used for a long time to estimate evolutionary distances [**?**]. However, it remains unknown how to extend these results to indels, which may be under higher selection pressure. Also, indels are commonly clustered, because a single evolutionary event likely causes an indel spanning multiple nucleotides.

Figure **??** shows the ideal results for both approaches. The lower, dashed line is the constant substitution rate, whereas the upper line is the error rate, counting each substitution and each indel as a single evolutionary event. As long as the indel rate ϕ is one order of magnitude smaller than the substitution rate of $\pi = 0.1$, all methods estimate π quite well.

Beyond that point, all methods become increasingly upwards biased, with ANDI growing fastest. Its estimations rise beyond the error rate, start varying heavily and fail because of missing anchors past an indel rate of $\phi \geq 0.256$. SPACED and KR show similar dynamics, but with smaller estimated rates than ANDI . COPHYLOG is surprisingly resistant to indels up to a rate of 0.128. Only for $\phi = 0.256$ and thus, a total error rate of $\pi + \phi = 0.356$, do its estimations become upwards biased.

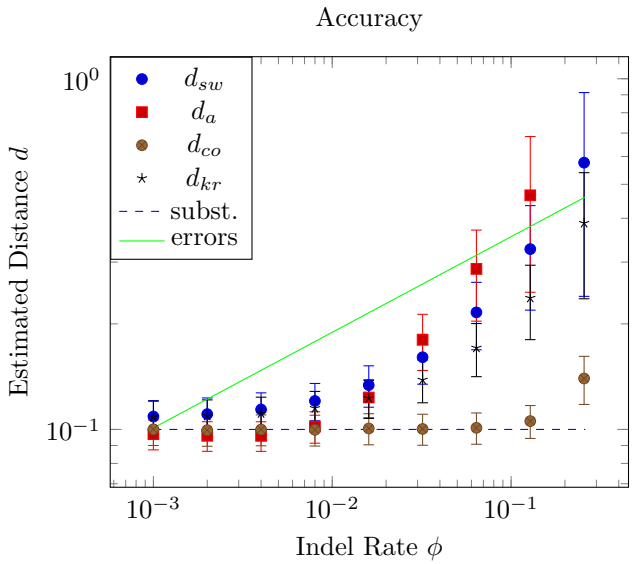


Figure 3.3: For each data point, one hundred sequence pairs with a distance of 0.1 and a certain indel rate were simulated. The mean and variance are plotted. Both theoretical distances substitutions and errors are shown as lines (continuous and dashed, respectively).

3.5 Recombination

When discussing the accuracy in Section **??** we assumed that substitutions are generated by a single Poisson process. In other words, the substitution rate π does not vary along or among the sequences. However, this is often not the case in real data because of *recombination* (i. e., crossover).

Recombination leads to variation in the substitution rate along a sequence. Figure **??** shows the local substitution rate within windows of 100 nucleotides along a recombined sequence of 1 kbp. A good distance estimator should be resistant to recombination.

As a test, two sequences with length 1 Mbp were simulated with a substitution rate of 0.1 using the tool MS [**?**]. Additionally a *population recombination rate* ρ ranging from 0.001 to 0.256 was introduced. Figure **??** shows the distances estimated by various methods for different levels of recombination. The distances computed by COPHYLOG, KR, and SPACED become downwards biased for increasing rates of recombination. ANDI is least affected by recombination. It rarely deviates more than 8% from the real distance and even gets better for higher rates of recombination. I suspect the reason for this is, it gets easier for highly clustered substitutions to find anchors in the flanking sequences.

3.6 Real Data

The ultimate test for alignment-free distance estimation is its application to real data. Unlike simulated sequences, real data is riddled with surprises like indels, recombination, and sequencing artifacts. In this section we explore the usability of the various distance methods when applied to three genomic data sets.

Escherichia Coli and *Shigella*

The ECO29 data set consists of 29 *Escherichia Coli*/*Shigella* genomes, which have previously been used for benchmarking distance methods [**?**, **?**]. On average, the genomes have a length of 4.9 Mbp. As a first surprise, they contain not only the standard nucleotides A, C, G, and T, but also R, D, N and even other characters in small numbers. These stand for groups of nucleotides: R is a purine (A or G), N is any nucleotide and

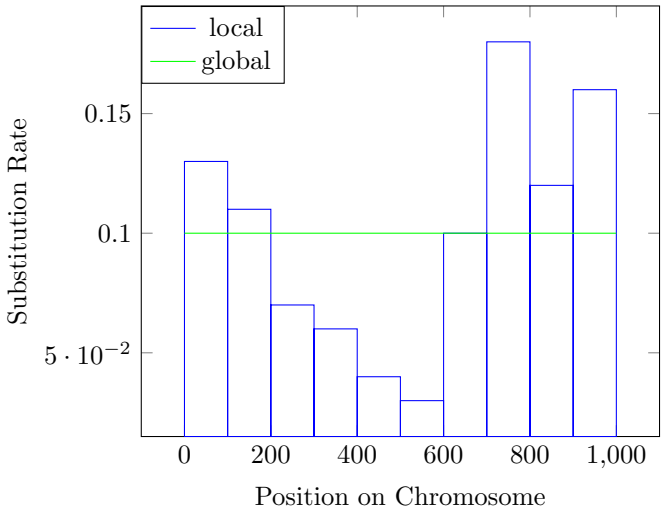


Figure 3.4: A chromosome of length 1000 was simulated with a global substitution rate of 0.1. An equal rate of recombination was introduced. This leads to fluctuations in the local substitution rate. The blue bars represent the local diversity within windows of 100 nucleotides.

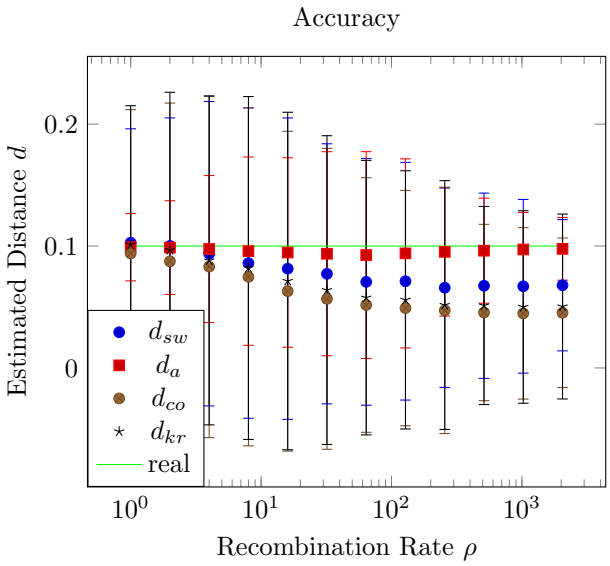


Figure 3.5: For each data point, one hundred sequence pairs with a distance of 0.1 and a certain recombination rate were simulated. The mean and standard deviation are plotted. Additionally, the straight line represents the simulated distance.

Table 3.1: Tree Metrics; All reported distances are with respect to MUGSY.

	ANDI	KR	COPHYLOG	SPACED
RSPR	1	6	3	6
branch score	0.001739	0.013654	0.009008	0.01415

D means any nucleotide but C. For some implementations these need to be filtered away. The complete FASTA file for the 29 genomes comprises of 138 MB.

Figure **??** shows the resulting phylogenies of four alignment-free distance measures as well as an alignment-based tree as reference. All trees were computed from the distance matrices using PHYLIP NEIGHBOR, RETREE and drawn with FIGTREE [**?**].

The visually worst result is computed by KR. A lot of its branch lengths differ noticeably from the reference tree by MUGSY. The three phylogenies by ANDI, COPHYLOG, and MUGSY are quite similar and nearly indistinguishable. SPACED fails to cluster four *E. Coli K12* strains together tightly.

These differences across the trees are now quantified using different metrics. First, RSPR is used to compute the topological difference between each alignment-free method and the reference tree [**?**]. ANDI has the smallest difference (1), followed by COPHYLOG (3). As expected, KR and SPACED have the worst scores (see Table **??**). The *branch score* distance also takes the length of branches into account [**?**]. Thus, a smaller branch score distance means, the length of two trees are more similar. Again, the tree by ANDI most closely resembles the reference, followed by COPHYLOG, KR, and SPACED, in this order.

The computation of the reference tree with MUGSY took 2 h, 49 min using 3 GB of memory on machine M1. The only method needing equally much memory is KR (see Figure **??**). But KR is one and a half orders of magnitude faster with an average runtime of 5 min, 23 s. Thus, it is even faster than the multithreaded SPACED, using eight cores. COPHYLOG is slightly faster than KR, but uses only 157 MB of **RAM!** (**RAM!**), making it the most memory-efficient tool. The fastest tool is ANDI , with just 100 s for the sequential and 27.7 s for the parallel case (eight threads).

Roseobacter

The genus of *Roseobacter* contains highly divergent bacteria, which makes them harder to compare than *E. coli*. A set of 32 *Roseobacter* genomes was recently used to evaluate the results of alignment-free distance estimators for diverse genomes [**?**]. A tree based on alignments of genes was used as a reference [**?**].

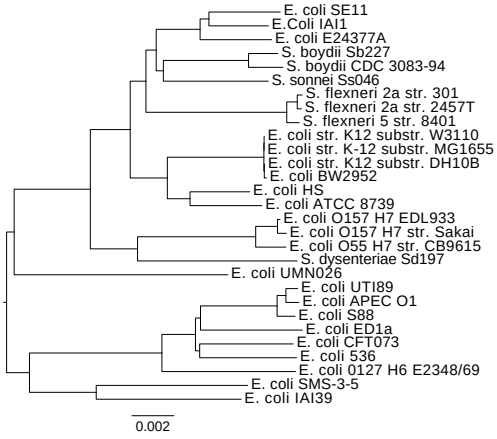
It was shown that with appropriate parameters (e. g., $k = 20$), SPACED could compute a tree with an RF-distance of 25 [**?**], making it the most accurate method evaluated. KR scored 46 and COPHYLOG 28 with $k = 28$. However, with default parameters, COPHYLOG only achieves 39, which is worse than the 33, we measured for ANDI .

The average evolutionary distance reported by ANDI for the *Roseobacter* genomes is one order of magnitude higher than for ECO29 (0.22 to 0.019). At the same time, the coverage (i. e., the amount of mapped homologous nucleotides) dropped from 0.765 to 0.046. This means, the result by ANDI is based on only 5% of the genome. It is interesting that 5% of a genome suffice to gain an answer as good—or bad—as with competing methods.

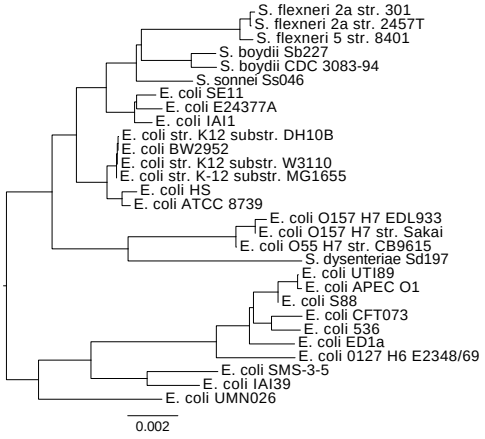
Streptococcus pneumoniae

The largest data set used in this thesis, PNEU3085, contains 3085 genomes of *Streptococcus pneumoniae* [**?**]. Each of these genomes is given as several contigs, amounting to 2.2 Mbp per genome and thus, 6.8 GB for the complete data set. As all of these genomes are compared pairwise, this results in more than 4 million comparisons (9 million, if asymmetric).

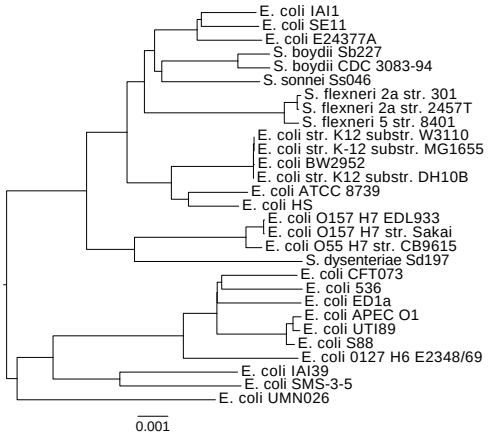
It is impossible to compute distances for this data set using KR and SPACED; both quickly exceed the available memory (256 GB) on machine M2. Thus, only the results for ANDI and COPHYLOG can be given here. Unfortunately, no reference tree exists or can be computed via an alignment; As this data set is



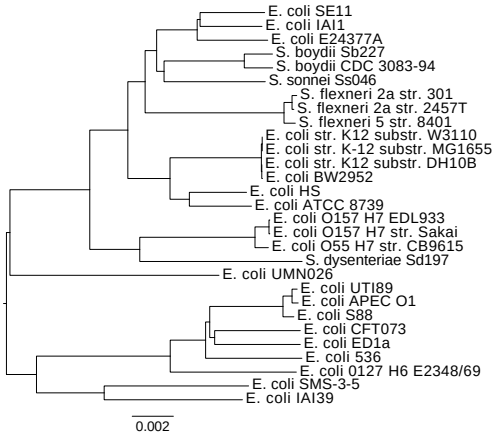
(a) ANDI



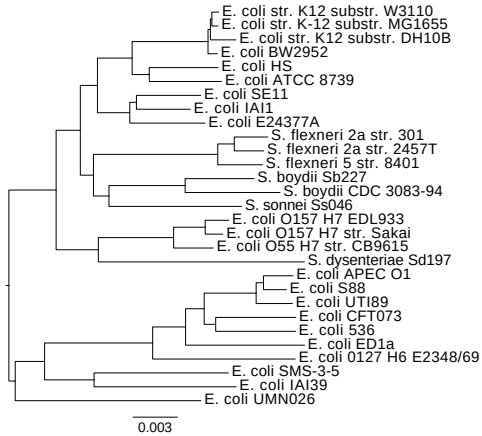
(b) KR



(c) COPHYLOG



(d) MUGSY



(e) SPACED

Figure 3.6: Phylogenies for the ECO29 data set.

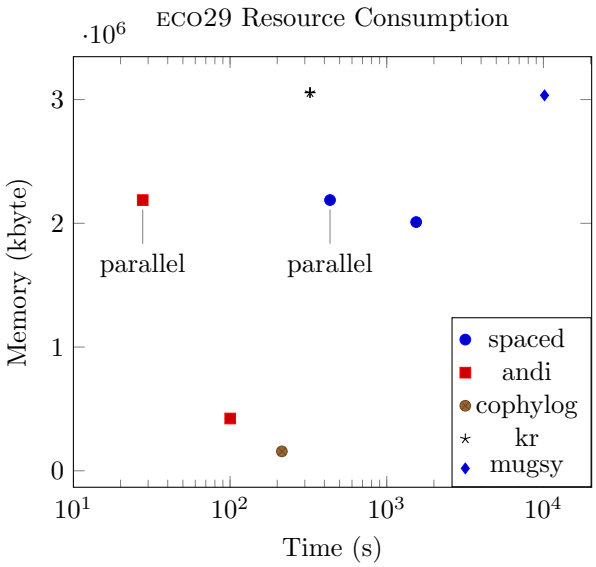


Figure 3.7: Resource consumption for the ECO29 test case. For all methods, except MUGSY, the means and variance of ten runs are shown.

roughly 100 times bigger than ECO29, it needs 100^2 times more pairwise genome comparisons. Thus, the runtime for MUGSY (2 h 49 min) would explode to approximately 3.2 years, which is impractical.

The figures on page **??** show the phylogenies based on the distances computed by ANDI and COPHYLOG. The most noticeable difference is the varying scale. The average distance computed by ANDI is 0.011 and 0.0057 for COPHYLOG. The RF-distance between the two trees is 4544. This may seem big, but is smaller than the average distance for two random trees of that size (6166) [**?**].

It took ANDI 6 h, 21 min and 10 GB of **RAM!** to compute the distance matrix on M2 with 32 threads. COPHYLOG is only single-threaded and ran for 36.5 days at just 2.3 GB. Even if COPHYLOG supported multi-threading, ANDI still is approximately four times faster.

3.7 Further Characteristics of ANDI

3.7.1 Threshold

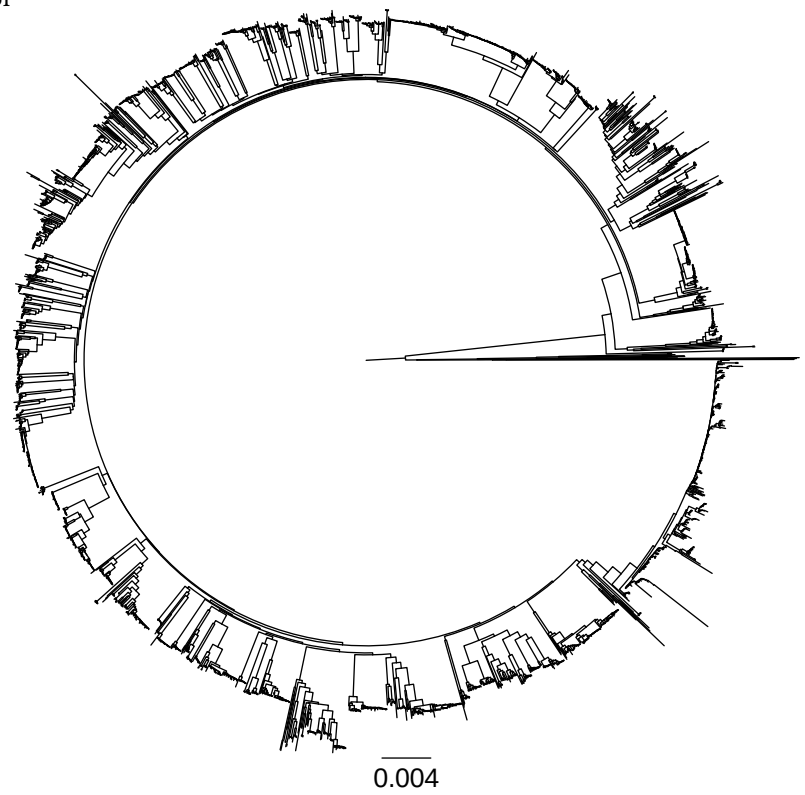
Recall from Section **??** that an *anchor* is a unique match of minimum length L. Since we are interested in anchors framing homologous regions, L should be picked so that random matches are unlikely. For this, another parameter p is needed, which represents the significance of an anchor pair.

$$p = 1 - P[\text{random pair}] \tag{3.1}$$

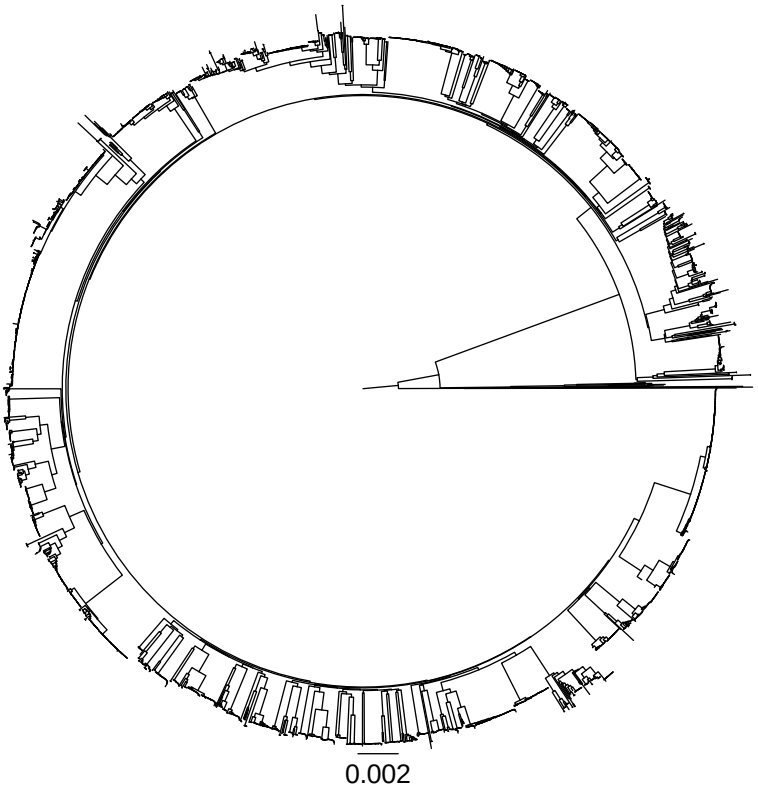
$$P[\text{random anchor}] = \sqrt{(1-p)} \tag{3.2}$$

The probability that an anchor was found by chance alone, in Equation **??**, depends on the length of the match. It is less likely for a long match to equal an arbitrary section in the subject than for a short match. The exact distribution of match lengths was described by [**?**]. For ANDI , a default p of 0.05 is picked. This results in a threshold L between 10 and 16, depending on the characteristics of the compared sequences. This is much lower than the average anchor length of 60, depending on the chosen data set (here ECO29, see Section **??**). Figure **??** displays the relationship between the threshold and the resulting distance.

(a) ANDI



(b) COPHYLOG



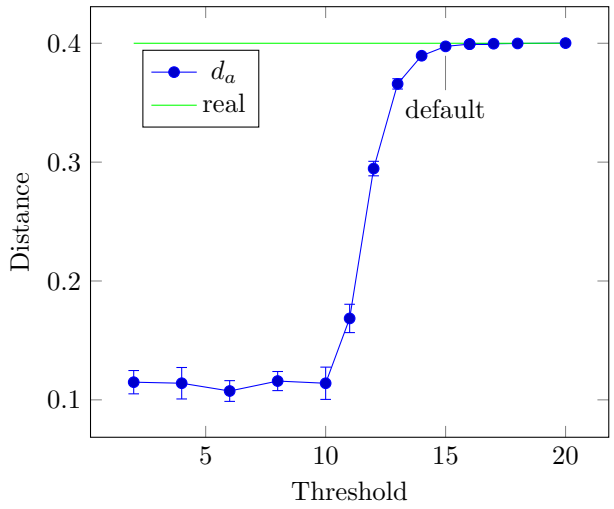


Figure 3.9: For each data point, ten sequence pairs with length 1 Mbp were simulated with a distance of 0.4. On the y-axis is the distance estimated by ANDI for the specific threshold. The default p-value of 0.05 equals a threshold of 15.

	left anchor						left anchor			
S:	A	A	G	T	A	–	G	C	T	T
Q:	A	A	G	T	A	A	G	C	T	T

Figure 3.10: This figure shows a worst case for the *anchor strategy*, where anchors are found, but are not equidistant and thus, do not form a proper pair. The gap »–« does not exist in the data but is shown here for improved clarity.

3.7.2 Complexity

The requirement for fast computation of the anchor distance is low algorithmic complexity and low memory usage. Recall from Chapter **??** that computing a match to a reference **ESA!**, takes time $O(m\sigma)$ where m is the length of the match and σ is the size of the alphabet. For our use case, Σ is the genomic alphabet, and hence, constant. Every nucleotide of the query is matched against the subject exactly once, leading to a runtime of $O(n)$.[2] In the worst case, every nucleotide is touched again for the computation of **SNP!**s. This still requires time $O(n)$ and $O(1)$ auxiliary working space.

The most time-consuming step is the creation of the **ESA!** for the subject. As shown in Chapter **??**, computing an **SA!** (**SA!**), **LCP!** (**LCP!**), **FVC!** (**FVC!**) and **RMQ!** (**RMQ!**) can be done in linear time, of which the **SACA!** (**SACA!**) takes longest, in practice. The memory requirement is $\Theta(n)$ for the **ESA!**.

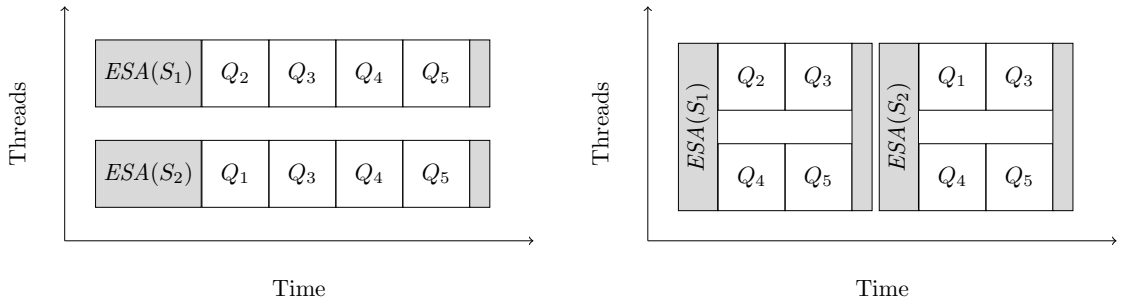
If more than just two sequences need to be compared, multiple queries can be streamed against the same **ESA!**. If k sequences are compared, streaming all queries against one subject takes $O(n)$ time for the **ESA!** construction (in theory) and $O(nk)$ time for comparison. With each sequence being a subject, computing the complete distance matrix is $O(nk^2)$ with $\Theta(n)$ working memory for the anchor distance, $O(nk)$ for the sequence data, and $O(k^2)$ for the matrix.

3.7.3 Worst Case Estimations

The `count_diff` function in Listing **??** computes a *Hamming distance*. This means it cannot detect indels. To protect against this, anchor pairs are required to be equidistant. This strategy leads to the following two problems.

[2] W.l.o.g. $|S| = |Q| = n$ is assumed.

Figure 3.11: The anchor pair frames a sequence of four nucleotides. As can be seen in the alignment, it contains two gaps. However, the Hamming distance does not see the gaps and instead counts three substitutions.



(a) Parallelization along subjects

(b) Parallelization along queries

Figure 3.12: The two subfigures show the two different possible modes of parallelization.

In the example shown in Figure **??**, the query contains one more character than the subject. When $d_{asym}(S, Q)$ is called, the first found anchor is AAGTA. Then the assumed substitution A is skipped and the second anchor is GCTT. Both anchors are unique and pass the threshold, which shall be 2, for the sake of this example. Unfortunately though, the anchors are not equidistant on both sequences and thus, no Hamming distance for the framed nucleotide(s) can be computed.

The alignment from Figure **??** has twice the previously described problem. The middle part, framed by the two anchors, is optimally aligned using two gaps. Unfortunately, the Hamming distance counts three substitutions instead. This way, indels, which cancel themselves out, could lead to great inaccuracies. The effects of this are evaluated in Section **??**.

3.7.4 Concurrency

Assume a comparison of five genome sequences. Then the calculation of $d_{asym}(1, 2)$ is independent of $d_{asym}(3, 2)$. So theoretically, they can be run in parallel; in fact, all comparisons for different subjects may be run in parallel. Comparisons against the same subject, however, have to await the precomputation of **ESA!** and end in its destruction.

At the current state of technology, multi-core processors are widely available. The number of processors p per machine ranges from two for smartphones, eight for standard home computers up to 64 for servers. Even though, p has been growing rapidly in the past years, it is usually still smaller than k.

If $p \leq k$ then the computation of the distance matrix is *embarrassingly parallel*. Two modes become apparent: computing multiple rows in parallel with each entry sequential or sequentially computing the entries of the rows in parallel (Figure **??**). The first mode computes multiple **ESA!**s in parallel and then streams all queries against them. Thus, it can be thought of parallelization along different subjects. This reduces the runtime to $O(nk^2/p)$, for $p < k$, at an increased memory usage of $O(nk + np + k^2)$.

The second mode is algorithmically more challenging, as it requires the **ESA!** to be built in parallel. But its advantage is that it holds only the **ESA!** for a single sequence in memory, instead of p, and thus, it uses less memory—$O(nk + n + k^2)$—which is identical to the sequential case. The runtime is likewise reduced to $O(nk^2/p)$ in theory, where parallelization of the **SACA!** has the biggest impact in practice.

more copy&paste

4 DevOps

ANDI is written in C/C++; mostly C99 with some parts in C++11. The sources are released on GITHUB as *free software* under the GNU GENERAL PUBLIC LICENSE VERSION 3 [?]. Prebundled packages using AUTOCONF are also available, with the latest release being v0.9 at the time of writing.

To prove correctness even under exceptional circumstances, the code was statically analyzed by the SCAN-BUILD utility from the LLVM framework [?].

4.1 Code Documentation

Every function in ANDI is documented using DOXYGEN style comments. To create the documentation run `make code-docs` in the main directory. You will then find the documentation under `./docs`, duh.

4.2 Unit Tests

The unit tests achieve a coverage of more than 80%. They are located in the ANDI repository under the `./test` directory. Because they require GLIB2, and a C++11 compiler, they are deactivated by default. To enable them, execute

```
~/andi % ./configure --enable-unit-tests
```

during the installation process. You can then verify the build via

```
~/andi % make check
```

The unit tests are also checked each time a commit is send to the official repository. This is done via TRAVISCI.[1] Thus, a warning is produced, when the builds fail, or the unit tests to not run successfully. Currently, the unit tests cover more than 80% of the code. This is computed via the TRAVIS builds and a service called COVERALLS.[2]

4.3 Known Issues

These minor issues are known. I will fix them, when I have time.

1. Stripping the filename from its path in join mode will not work properly on Windows because of the different path separator.
2. I currently do not check for integer multiplication overflows in mallocs. To fix this, I might be using *reallocarray* from BSD.

4.4 Creating a Release

A release should be a stable version of ANDI with significant improvements over the last version. For bug fixes, dotdot release may be used.

Once ANDI is matured, the new features implemented, and all tests were run, a new release can be created. First, bump the version number in *configure.ac*. Commit that change in git, and tag this commit with

[1] https://travis-ci.org/EvolBioInf/andi
[2] https://coveralls.io/r/EvolBioInf/andi

vX.y. Create another commit, where you set the version number to the next release (e. g., *vX.z-beta*). This assures, there is only one commit and build with that specific version. Now return to the previous commit `git checkout vX.y`.

Assert that ANDI is ready for packaging with AUTOCONF.

```
~ % make distcheck
make dist-gzip am__post_remove_distdir='@:'
make[1]: Entering directory `/home/kloetzl/Projects/andi'
if test -d "andi-0.9.1-beta"; then find "andi-0.9.1-beta" -type d ! -
    perm -200 -exec chmod u+w {}';' && rm -rf "andi-0.9.1-beta" || {
    sleep 5 && rm -rf "andi-0.9.1-beta"; }; else :; fi
test -d "andi-0.9.1-beta" || mkdir "andi-0.9.1-beta"
 (cd src && make top_distdir=../andi-0.9.1-beta distdir=../andi-0.9.1-
     beta/src \
    am__remove_distdir=: am__skip_length_check=: am__skip_mode_fix=:
       distdir)

... Loads of output ...

================================================
andi-0.9.1-beta archives ready for distribution:
andi-0.9.1-beta.tar.gz
================================================
```

If the command does not successfully builds, no tarballs will be created. This is when you really have to dive into the internals of AUTOCONF and AUTOMAKE. Good luck with that!

5 Notation

Cleanup

SA suffix array
SACA suffix array construction algorithm
LCP longest common prefix
ESA enhanced suffix array
CSA compressed suffix array
DNA deoxyribonucleic acid
SNP single nucleotide polymorphism
RMQ range minimum query
FVC first variant character
CPU central processing unit
GPU graphics processing unit
RAM random access memory
PRAM parallel random access memory
CREW concurrent read exclusive write
CLD child array