Efficient Relaxed Search in Hierarchically-Clustered Sequence Datasets

KAI C. BADER, Technische Universität München
MIKHAIL J. ATALLAH, Purdue University
CHRISTIAN GROTHOFF, Technische Universität München

This paper presents a new algorithm for finding oligonucleotide signatures that are specific and sensitive for organisms or groups of organisms in large-scale sequence datasets. We assume that the organisms have been organized in a hierarchy, for example a phylogenetic tree. The resulting signatures, binding sites for primers and probes, match the maximum possible number of organisms in the target group while having at most k matches outside of the target group.

The key step in the algorithm is the use of the Lowest Common Ancestor (LCA) to search the organism hierarchy; this allows to solve the combinatorial problem in almost linear time (empirically observed). The presented algorithm improves performance by several orders of magnitude in terms of both memory consumption and runtime when compared to the best-known previous algorithms while giving identical, exact solutions.

This paper gives a formal description of the algorithm, discusses details of our concrete, publicly available implementation and presents the results from our performance evaluation.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; J.3 [**Life and Medical Sciences**]: Biology and Genetics

General Terms: Applied algorithms, data mining

Additional Key Words and Phrases: oligonucleotide, probe design, primer design, signature, phylogenetic, tree, ribosomal rna

ACM Reference Format:
Bader, K. C., Atallah, M. J., and Grothoff, C. 2012. Efficient Relaxed Search in Hierarchically-Clustered Sequence Datasets. ACM J. Exp. Algor. 0, 0, Article 0 (2012), 19 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Molecular diagnostic techniques, which are applying polymerase chain reaction (PCR) [Bartlett and Stirling 2003] or RNA/DNA hybridization [Amann et al. 1995; Stahl et al. 1988], are becoming a standard in various fields of life sciences and medicine. They rely on oligonucleotide primers and probes, short (15–25 bases) sub-

Portions of this work were supported by the Bayerische Forschungsstiftung AZ 767-07; the Deutsche Forschungsgemeinschaft ENP GR 3688/1-1; the National Science Foundation Grants CNS-0915436, CNS-0913875, Science and Technology Center CCF-0939370; by an NPRP grant from the Qatar National Research Fund; by Grant FA9550-09-1-0223 from the Air Force Office of Scientific Research; and by sponsors of the Center for Education and Research in Information Assurance and Security. The statements made herein are solely the responsibility of the authors.
Author's addresses: Kai C. Bader, kb@tum.de, Technische Universität München, München, Germany; Mikhail J. Atallah, mja@cs.purdue.edu, Purdue University, Computer Science Department, West Lafayette, IN 47907; Christian Grothoff, grothoff@in.tum.de, Technische Universität München, München, Germany.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

sequences of DNA or RNA. These subsequences bind to longer sequences from a biological sample to start the desired biochemical reaction. Binding sites on the samples, the complement of the primers or probes, are called oligonucleotide signatures (hereafter simply referred to as signatures). In order to interact only with sequences from the target organism, they must be specific to the target organism or group of organisms.

A source for such primers and probes can be curated databases like probeBase [Loy et al. 2007], or in many cases, the design "by hand". Given the size of modern sequence datasets, software tools are necessary to design new or re-evaluate already published candidates before testing in the wet lab. In this paper we present an efficient algorithm for the comprehensive *in silico* search for good signature candidates (Figure 1). Maintainers of gene and genome databases, such as SILVA, RDP, or Greengenes [Pruesse et al. 2007; Cole et al. 2009; DeSantis et al. 2006] could use it to precompute and offer a collection of signature candidates along with their datasets. But it also is useful for end-users with custom sequence collections based on projects like FunGene[1].

For applications in this domain, our algorithm assumes that the organisms are hierarchically clustered. Clusterings could be based on any kind of classification where inner nodes represent related groups. An example is a phylogenetic tree: Inner nodes represent derived evolutionary relationships between groups of organisms, and the individual organisms correspond to the leaves. Phylogenetic trees, if not supplied with sequence datasets, can be computed using third-party tools such as FastTree or RAxML [Price et al. 2010; Stamatakis 2006].

The other main input is a bipartite graph (Figure 1, center) that relates signatures to matched sequences (organisms). A relation means that a signature is present in a sequence. Such a bipartite graph is easily computed using existing search index tools that construct suffix trees over sequence data. Signatures are extracted by traversing the tree until a certain (length) constraint is met, and references to the sequences can be found in the underlying nodes.

It should be noted that depending on the target group and the available sequence data, there may not be a perfect match; a perfect match would be a signature that matches all target sequences and has no matches outside of the target group (no false-positives). Thus, we are interested in algorithms that support relaxed search conditions; the algorithm should minimize the number of target sequences that are not matched (false-negatives) while allowing for at most k false-positives (where k is typically a small number). In practice, inaccuracies in the available sequence data and additional constraints (such as melting point restrictions) complicate the situation further. However, many of these issues have been addressed in prior work [Bader et al. 2011] and shall thus remain outside of the scope of this paper. In this work we concentrate on the central algorithm for the search for good signature candidates, hereinafter referred to as CaSSiS-LCA.

We will use the following formalism to present and discuss our algorithm. Let $G = (U, V, E)$ be a bipartite graph where edges $(u, v) \in E$ represent that a signature $u \in U$ matches an organism $v \in V$. Furthermore, let T be a tree with leaves in V (for example, T might represent a phylogenetic tree). Furthermore, let $D(t)$ be the set of all descendants of $t \in T$ and $P(t)$ the result set of signatures for t. Then, this paper presents an $O(k|U|\log|V| + k|V| + |E|)$ time algorithm (detailed analysis in Section 3.3) which determines for all elements $t \in T$ those element(s) $u \in U$ which maximize the number of edges $(u, v) \in E$ with $v \in D(t)$ while not having more than k edges $(u, v') \in E$ with $v \notin D(t)$. For each t, the resulting signatures are stored in $P(t)$. Note that in

[1]`http://fungene.cme.msu.edu/`

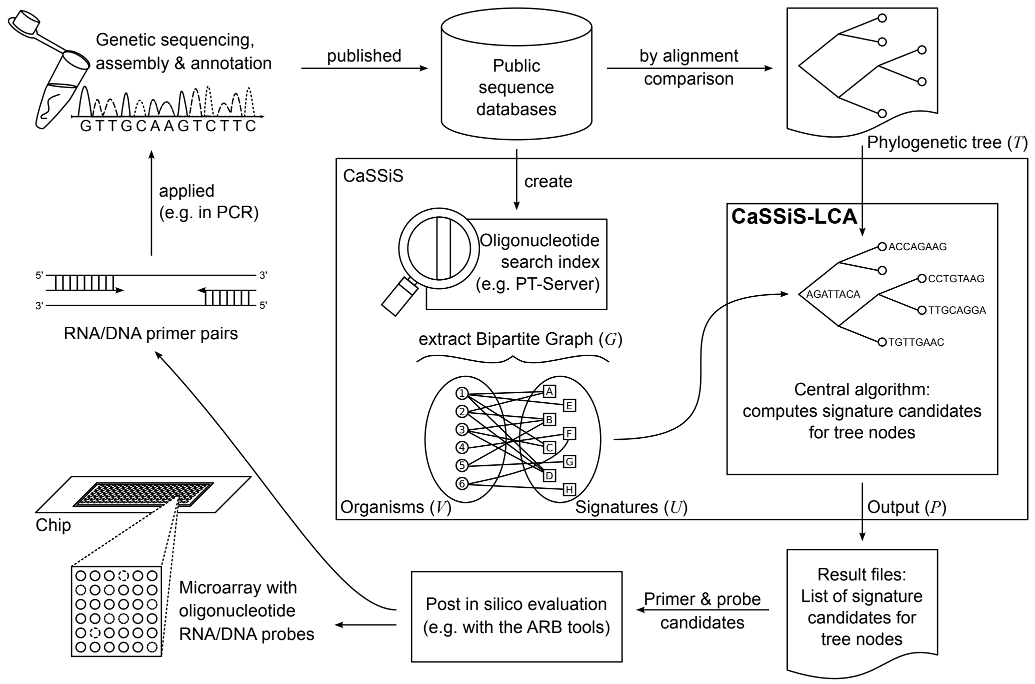

Genetic sequencing, assembly & annotation
GTTGCAAGTCTTC
published
Public sequence databases
by alignment comparison
Phylogenetic tree (T)
applied (e.g. in PCR)
CaSSiS
create
Oligonucleotide search index (e.g. PT-Server)
CaSSiS-LCA
ACCAGAAG
AGATTACA
CCTGTAAG
TTGCAGGA
TGTTGAAC
RNA/DNA primer pairs
extract Bipartite Graph (G)
Central algorithm: computes signature candidates for tree nodes
Organisms (V)
Signatures (U)
Output (P)
Chip
Microarray with oligonucleotide RNA/DNA probes
Post in silico evaluation (e.g. with the ARB tools)
Primer & probe candidates
Result files: List of signature candidates for tree nodes

Fig. 1: Schematic of the primer and probe design pipeline in which CaSSiS is embedded. The input data for the new algorithm CaSSiS-LCA comes from public sequence and phylogenetic tree databases. The resulting signatures can be used as a template for new RNA/DNA primers and probes, for example to provide diagnostic microarrays.

practice $|E|$ is several orders of magnitude larger than $k|U|\log|V|$ and thus the runtime is practically linear in the size of the input.

Following [Bader et al. 2011], we will call edges $(u, v) \in E$ with $v \in D(t)$ *ingroup* matches for group t and edges $(u, v') \in E$ with $v \notin D(t)$ *outgroup* matches for group t. The bound k is the maximum number of outgroup matches that can be tolerated. If the resulting signature u is used for diagnostics, outgroup matches would result in false-positive tests. The goal of the algorithm is to maximize the number of ingroup matches. If there exist organisms $v \in D(t)$ where $(u, v) \notin E$, this would result in false-negatives tests when using signature u to test for group t. In other words, the presented algorithm finds for each group of organisms $t \in T$ all of those signatures $u \in U$ that have less than k false-positives and minimize false-negatives.

The remainder of this paper is structured as follows. We review related work in Section 2. Our algorithm is presented in Section 3. Details about our implementation and its performance are given in Section 4. We do not include specific signatures as the biological results are identical to those already presented in [Bader et al. 2011].

2. RELATED WORK

Nowadays the use of primers and probes is widespread in the field of medical diagnostics. In previous work we have shown that computational methods can find typical representatives that are applied in this field [Bader et al. 2011]. An example is *EUB338*, a domain-specific probe used for the detection of organisms classified as "Bacteria" [Amann and Fuchs 2008]. Our algorithm not only found the signature corresponding to EUB338 but even presented a signature with a higher coverage (its position was shifted by one base compared to EUB338) [Bader et al. 2011].

The *in silico* search for oligonucleotides is provided by various tools. Most of them are specialized in either primer design [Duitama et al. 2009; Peterlongo et al. 2009; Ficetola et al. 2010; Riaz et al. 2011] for PCR applications, or the design of probes [Chung et al. 2005; Feng and Tillier 2007] that could be applied in DNA microarrays. But also more generic approaches exist that try to identify signature sites [Ludwig et al. 2004; Lee et al. 2010].

For various reasons we found these tools unsuitable for the comprehensive computation of hierarchically clustered sequence datasets. They are typically limited to processing data sets of a few thousand gene sequences or a few genomes due to excessive memory or time requirements [Bader et al. 2011]. In most cases, oligonucleotides are computed only for one predefined set of targets and non-targets per run. Another limit is the lack of relaxed non-heuristic search methods, which gain importance when processing large datasets [Bader et al. 2011].

Prior to this paper, the only algorithms known to us capable of doing a comprehensive non-heuristic signature computation based on large hierarchically linked gene and genome sequence datasets were Insignia [Phillippy et al. 2007] and the first CaSSiS implementation, hereinafter referred to as CaSSiS-BGRT [Bader et al. 2011].

2.1. Insignia

Insignia is a web application developed and maintained by the Center for Bioinformatics and Computational Biology at the University of Maryland, USA. It currently (March 2012) contains 13,928 genomic organism sequences (11,274 viruses/phages and 2,653 non-viruses).

Insignia consists of two pipelines. The first "match pipeline" is used to pre-compute "match cover" arrays M for every pair of organisms. For example, for $v_1, v_2 \in V$ and $v_1 \neq v_2$, the match cover array $M(v_1, v_2)$ contains the positions and lengths of all sequence regions of v_1 that are also present at one or more positions on v_2. To find common regions for an organism pair, Insignia uses MUMmer [Kurtz et al. 2004] to build a suffix-tree based search index. Signatures of a defined length matching v_1 and v_2 are extracted, merged if their positions on v_1 overlap, and added as regions to the match cover.

A match cover consists of integers, i.e. position and length pairs on a reference sequence. The number of pairs in a match cover is bound by the sequence length l. To process $\approx$ 80 billion nucleotides from NCBI RefSeq genome database [2], the first pipeline had to be distributed across a a 192-node cluster [Phillippy et al. 2009]. The authors did not provide information about the actual runtime of the algorithm. The memory consumption for the match cover M for 300 organisms is reported to be only $\approx$ 2 GB [Phillippy et al. 2007].

After pre-computing the match cover, the second "signature pipeline" is triggered over Insignia's web interface. The signature pipeline computes regions (i.e. one or more overlapping signatures) shared between a user-defined group of target organisms $V' \subseteq V$ which must also be absent in the background $V'' = V \setminus V'$ [Phillippy et al. 2007]. One target organism $v' \in V'$ is designated as the reference organism and used to visualize the result.

The signature pipeline consists of three steps. In a first step, an intersection $I_{v'} := \bigcap_{v_t \in V'} M(v', v_t)$ of the match cover structures from v' with the other target organisms from V' is created. It contains only sequence regions on v' that are shared by all targets

[2]`http://www.ncbi.nlm.nih.gov/RefSeq/`

V'. In a second step, a union $U_{v'} := \bigcup_{v_b \in V''} M(v', v_b)$ of the match cover structures from v' with all background organisms V'' is created. $U_{v'}$ thus contains all regions on v' that match one or more organisms from the background V''. In the last step the regions from $I_{v'}$ are compared to the ones in $U_{v'}$ to find possible signature candidates. Valid signatures for the targets V' have to completely lie within a region on $I_{v'}$ and must not entirely lie within a region on $U_{v'}$.

All operations in the signature pipeline have (practically) linear time complexity in the size of the match cover [Phillippy et al. 2007]. The match cover intersection $I_{v'}$ for a target group V' (and a reference organism v') has a time complexity of $O(|I_{v'}| \log |V'|)$. The $\log |V'|$ factor can be treated as a constant due to the bounded number of genomes [Phillippy et al. 2007]. A single query takes on average one minute to process [Phillippy et al. 2007].

The final output of Insignia is a list of regions consisting of overlapping k-mer signatures. Insignia was primarily designed to process single queries (single targets or groups of target sequences) and not for handling deep hierarchies. In contrast to the work presented in this paper, Insignia is only able to report k-mer signatures perfectly matching the whole target group without allowing non-target matches. However, for many reasonable groups of organisms such perfect signatures often simply do not exist. In our test datasets, only 55% of the organisms and 14% of all groups were perfectly covered [Bader et al. 2011, Section 3.5] by one or more signatures. Furthermore, Insignia can also not be used to find signatures with small mismatches to the target sequences (which is useful to tolerate sequencing errors) or to enforce larger Hamming distances to non-target (background) organisms. CaSSiS and the improvements over CaSSiS that are presented in this paper address these shortcomings.

2.2. CaSSiS-BGRT

The CaSSiS-BGRT [Bader et al. 2011] algorithm and the CaSSiS-LCA algorithm presented in this paper use the same input sources and provide the same outputs. Specifically, both approaches use the ARB PT-Server [Ludwig et al. 2004] to construct a bipartite graph that matches signature candidates to organisms. ARB first generates all possible signatures of the specified length and then matches them (using a suffix trie) against the sequences of the organisms. The PT-Server supports approximate matching, for example to compensate for sequencing errors in the database. The algorithms then process the resulting data stream and generate a map P which contains for each $t \in T$ a set of promising signature candidates.

The two approaches differ in the central algorithm which searches and evaluates signature candidates. Given a phylogenetic tree and the bipartite graph, CaSSiS-BGRT uses a new data structure, the bipartite graph representation tree (BGRT), to process more than 460,000 gene sequences (660M nucleotides, matched without mismatches or outgroup hits) in about 132h on an Intel Core i7 with 24 GB of system memory [Bader et al. 2011]. The algorithm employed by CaSSiS has worst-case complexity $O(|M| \cdot |V|)$ time where M is the size of the BGRT (which is in turn bounded by $|U|$, the number of edges in the bipartite graph) and $|V|$ is the set of all organisms. CaSSiS-BGRT uses a time-memory trade-off to implement a bounded search to significantly reduce the execution time in practice; however, as a result, CaSSiS-BGRT requires $O(d \cdot |M|)$ memory where d is the depth of the (phylogenetic) tree T.

While 132h may seem sufficient to find signatures for all sequences and sequence groups of interest, SSURef 102 only contains long (> 900 nt) annotated aligned SSU rRNA sequences and not full genomes. CaSSiS-BGRT cannot be expected to process contemporary data sets containing full genomes as memory consumption is linear in the number of nucleotides and 700 million nucleotides already require about 16 GB of RAM. For comparison, a human genome has 3.3 billion nucleotides and ideally a signature search should to consider all available sequence data for all organisms. The CaSSiS-LCA algorithm presented in this paper significantly outperforms CaSSiS-BGRT both in terms of memory and time complexity and is thus able to process full genomes.

3. OUR ALGORITHM

In this section we present our new algorithm CaSSiS-LCA. We build up to the full-featured algorithm in three steps to introduce each of the key ideas separately and to properly highlight how the algorithm handles the different cases.

The bipartite graph $G = (U, V, E)$ has the key property that in practice we can expect there to be a relatively small number of organisms in V (hundreds of thousands) and many more signatures U (billions) and even more edges (Figure 2). Thus it is impractical to load E (or even U) into main memory at any given time. Existing tools that generate signature candidates and match them against organisms can efficiently create E in the form of a data stream, giving all of the tuples $(u, v) \in E$ for a given $u \in U$ in a single contiguous block in the overall stream. The basic philosophy of our algorithm is thus to do stream processing [Bader et al. 2010] over a stream that represents the bipartite graph. Each round of the algorithm is given a $u \in U$ and the set $S_u \subseteq V$ of all organisms $v \in S_u$ that match signature candidate u. Our stream processing algorithm must then decide to keep u in a preliminary result set or discard u for good.

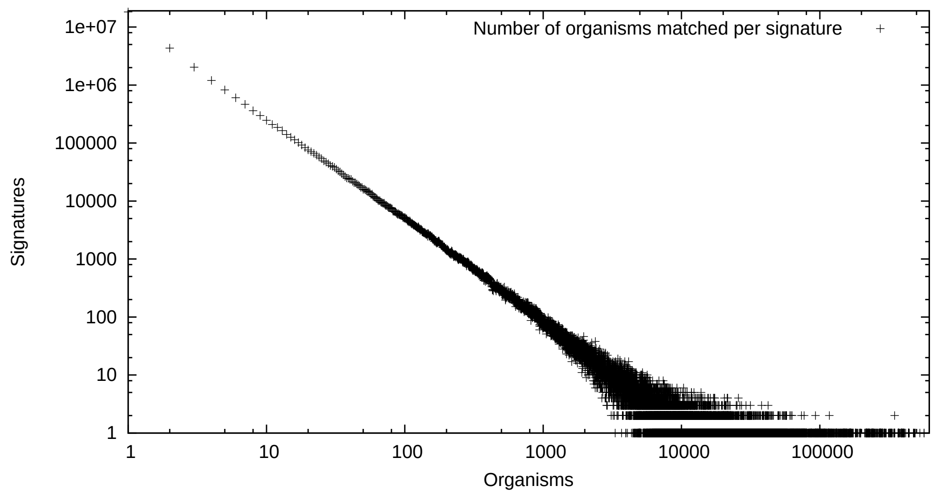


Fig. 2: This figure shows the number of signatures that reference a certain number of organisms. It is based on the bipartite graph of the complete SSURef 108 dataset ($618, 442$ organisms, $31, 976, 771$ signatures). 57% of the signatures match a single organism, only 11% match 10 or more organisms. Both axis use a logarithmic scale.

Prior to the main algorithms, we always perform some basic precomputations. First, we number the organisms sequentially in the tree T from left to right, thus (without loss of generality) $v \in \mathbb{N}$. Second, we precompute the sparse tables necessary for computing lowest common ancestors in T. This precomputation can be done in $O(|V|)$ time [Bender and Farach-Colton 2000; Fischer and Heun 2006; Gabow et al. 1984].

Henceforth, we can compute the Lowest Common Ancestor (LCA) of v and v' (denoted by $LCA(v, v')$) for $v, v' \in T$ in $O(1)$ time.

We will present the new algorithm in three consecutive steps. The simplest algorithm, presented in Section 3.1, is only considering perfect matches. A more relaxed algorithm which allows partial matches is given in Section 3.2. The actual algorithm which additionally allows outgroup matches follows in Section 3.3. A list of notations used in the three algorithms is shown in Table I.

$C(t)$	Child nodes of node $t \in T$
$D(t)$	Set of all descendants of $t \in T$
E	Edges $(u \in U, v \in V)$ in the bipartite graph
G	Bipartite graph (U, V, E)
indexof$_{min}(e, S)$	Returns the index of element $e \in S$ in the array S, if $e \notin S$, the index e would have had if e was in S minus one is returned
indexof$_{max}(e, S)$	Returns the index of element $e \in S$ in the array S, if $e \notin S$, the index e would have had if e was in S is returned
$LCA(v, v')$	Lowest Common Ancestor of $v \in V$ and $v' \in V$
$P(t)$	Result set for tree node $t \in T$
$P(t, k)$	Result set for tree node $t \in T$ and $k \in \mathbb{N}$ outgroup hits
S_u	Set of sequences $S_u \subset V$ that are matched by a signature u
sort S	Returns a sorted array with the elements from S
T	(Phylogenetic) Tree
U	Signature set
V	Sequence set

Table I: Notations that are used in the following three algorithms.

3.1. Perfect Match Algorithm

We begin our exposition with an algorithm for the simple case of finding a probe that provides perfect group coverage, that is, we are only interested in finding probes that match all sequences in the target group and have no outgroup hits.

Figure 3 illustrates the two key cases the perfect match algorithm considers. First, the common case that the organisms matched by the signature do not correspond perfectly to any group, and second the desired case that the signature corresponds exactly to a particular group, which LCA can identify in $O(1)$ time.

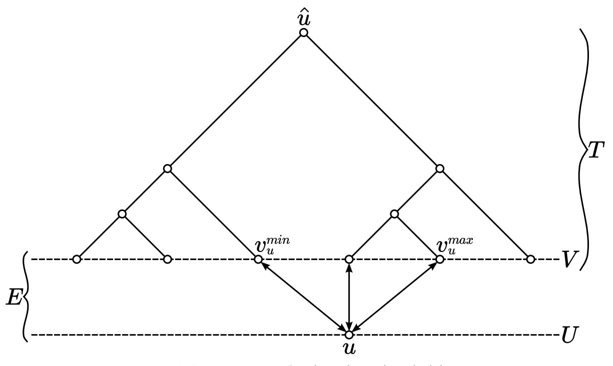
Algorithm 1 can then be used to find perfect matches for each organism or group of organisms (i.e. relation P). The key idea here is to use LCA to quickly determine the only $t \in T$ that *might* be matched perfectly by a given signature u, and then to use arithmetic to determine if u matches all descendants $D(t)$. For this algorithm, we need to precompute the number of descendants $|D(t)|$ for all $t \in T$ (which is trivial to do in $O(|V|)$ time). For determining the LCA $\hat{u}$, the minimum and maximum organism identifiers v_u^{min} and v_u^{max} are fetched and used. This can be done in $O(1)$ time in the sorted list S_u. Although sorting might be overdoing to find the minimum and maximum identifiers, it was used for consistency to the third (implemented) algorithm where it becomes mandatory.

The complexity of the computation given in Algorithm 1 is $O(|U| + |E|)$ ($|E|$ from sorting arrays of integers with a total of $|E|$ entries in linear time, $|U|$ from processing each signature u). Together with the precomputation, the overall complexity of this first algorithm is thus $O(|V| + |E| + |U|)$ time.

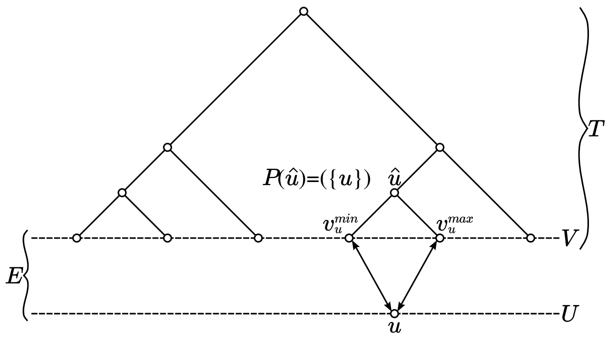
ALGORITHM 1: This algorithm computes a relation P that maps for each organism or group of organisms $t \in T$ to signature that perfectly cover all organisms in $D(t)$ (with no false-positives).

ALGORITHM: Perfect Matching

```
Input: $G(U, V, E)$, $T$, $LCA(v, v')$, $|D(t)|$
Output: $P : T \rightarrow \mathcal{P}(U)$
for $u \in U$ do
    $P(u) \leftarrow \emptyset$;
end
for $u \in U$ do
    $S_u \leftarrow$ sort $\{v_u | (u, v_u) \in E\}$;
    $v_u^{min} \leftarrow$ min $S_u$ ;
    $v_u^{max} \leftarrow$ max $S_u$ ;
    $\hat{u} \leftarrow LCA(v_u^{min}, v_u^{max})$;
    if $|S_u| = |D(\hat{u})|$ then
        $P(\hat{u}) \leftarrow P(\hat{u}) \cup \{u\}$;
    end
end
```



(a) No match. $|S_u| \neq |D(\hat{u})|$



(b) Perfect match. $|S_u| = |D(\hat{u})|$

Fig. 3: Illustration of Algorithm 1. In Case 3a the set size $|S_u| = 3$ is smaller than $|D(\hat{u})| = 6$, so the candidate u is not a perfect probe for any sequence or sequence group. Case 3b shows a probe that would be a perfect match ($|S_u| = 2 = |D(\hat{u})|$).

3.2. Partial Group Coverage

The second version of the algorithm will now relax the constraint that the group coverage must be perfect. Instead, we will permit that some organisms in the group are not covered by the signature. The goal of the algorithm is to find those signatures that provide the maximum group coverage (minimizing false-negatives) while not allowing any outgroup hits (no false-positives).

Let $r \in T$ be the root of the tree T and let $C(t) \subset T$ denote the children of $t \in T$. Algorithm 2 can then be used to find those signatures that maximize group coverage (with no outgroup hits) in $O(|U| + |E|)$ time. The result is stored in a map P which maps each $t \in T$ to a pair consisting of the set of signatures $\mathcal{U} \subseteq U$ and the number of organisms matched in the target group by all $u \in \mathcal{U}$.

ALGORITHM 2: This algorithm computes a relation P that maps for each organism or group of organisms $t \in T$ to the set of signatures that provide maximum coverage of the organisms in $D(t)$ (with no false-positives). The algorithm runs in $O(|U|+|V|+|E|)$ time as we can use bucket sort to sort in $O(|E|)$ time and Procedure PropagateUp adds $O(|V|)$ time.

ALGORITHM: Partial Group Coverage

```
Input: $G(U, V, E)$, $T$, $LCA(v, v')$
Output: $P : T \to (\mathcal{P}(U), \mathbb{N})$
1  for $u \in U$ do
2      $P(u) = (\emptyset, 0)$;
3  end
4  for $u \in U$ do
5      $S_u \leftarrow$ sort $\{v_u | (u, v_u) \in E\}$;
6      $v_u^{min} \leftarrow$ min $S_u$ ;
7      $v_u^{max} \leftarrow$ max $S_u$ ;
8      $\hat{u} \leftarrow LCA(v_u^{min}, v_u^{max})$;
9      $(\mathcal{U}, n') \leftarrow P(\hat{u})$;
10     $n \leftarrow |S_u|$;
11     if $n > n'$ then
12         $P(\hat{u}) \leftarrow (\{u\}, n)$;
13     end
14     if $n = n'$ then
15         $P(\hat{u}) \leftarrow (\mathcal{U} \cup \{u\}, n')$;
16     end
17 end
18 PropagateUp ($T$, $r$, $P$);
```

A key step in Algorithm 2 is the propagation of results up the tree in Procedure PropagateUp. This step exploits the fact that the parent p' of a node $p \in T$ always represents a superset of organisms and thus a probe that has no outgroup hits for p will also have no outgroup hits for p'. A suitable probe candidate for p can therefore also be a good candidate for p'. Thus, Procedure PropagateUp is needed to ensure that probes which provide partial group coverage are found.

Note that for partial group coverage, the computation of $|D(t)|$ is no longer required.

Figure 4 illustrates the key steps in Algorithm 2. Given a signature u, the algorithm first determines v_u^{min} and v_u^{max}, then determines $\hat{u}$ using LCA, associates the signature u with $|S| = 2$ ingroup hits with $P(\hat{u})$, and finally (assuming there were no other, better signatures found in U) propagates the signature u to the ancestors of $\hat{u}$.

Procedure PropagateUp(T, p, P): Helper function to propagate good matches up the tree using depth first traversal (in $O(|V|)$ time).

```
Input: $T, p, P : T \to (\mathcal{P}(U), \mathbb{N})$
Output: Updated $P : T \to (\mathcal{P}(U), \mathbb{N})$
foreach $c \in C(p)$ do
    PropagateUp ($T$, $c$, $P$);
    if $p \neq r$ then
        $p' \leftarrow$ parent ($p$);
        $(\mathcal{U}, n) \leftarrow P(p)$;
        $(\mathcal{U}', n') \leftarrow P(p')$;
        if $n > n'$ then
            $P(p') \leftarrow (\mathcal{U}, n)$;
        end
        if $n = n'$ then
            $P(p') \leftarrow (\mathcal{U} \cup \mathcal{U}', n')$;
        end
    end
end
```

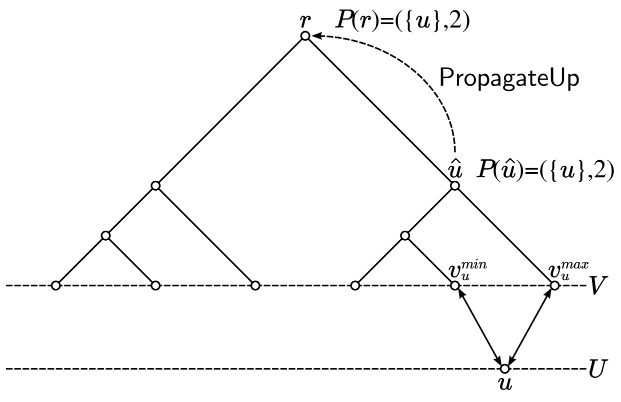


Fig. 4: Illustration of Algorithm 2. The coverage of the probe u is $|S_u| = 2$. It is added at the node $\hat{u}$. Note that the PropagateUp step happens once (for each $t \in T$) at the end of the algorithm and not after each $u \in U$.

3.3. Allowing at most k Outgroup Hits

Finally, we present the complete CaSSiS-LCA algorithm that computes for each $t \in T$ those signatures $u \in U$ that maximize the number of matched target organisms $v \in D(t)$, while not matching more than k organisms $v' \notin D(T)$. More precisely, our algorithm computes for each $i \in \{0, \dots, k\}$ and each $t \in T$ the set of signatures $u \in U$ that have exactly i outgroup hits and the maximum number of ingroup hits.

Before the main algorithm, we precompute for each $t \in T$ its leftmost and rightmost leaf in the subtree rooted at t, which we will refer to as the border $B(t) = (v_t^{min}, v_t^{max})$ (where $v_t^{min} := min(D(t))$ and $v_t^{max} := max(D(t))$). It is trivial to do this precomputation in $O(|V|)$ time.

Given this, Algorithm 3 computes P, a mapping from pairs $(T, \{0, \dots, k\})$ (representing a target organism or target group of organisms and a number of outgroup hits) to a pair $(\mathcal{P}(U), \mathbb{N})$ consisting of a set of signatures and the number of organisms

ALGORITHM 3: This algorithm computes a relation P that maps for each organism or group of organisms $t \in T$ to the set of signatures that provide maximum coverage of the organisms in $D(t)$ (with no false-positives). The algorithm runs in $O(k|U|\log|V| + k|V| + |E|)$ time. The algorithm requires $O(|V|)$ space for the LCA data structure and can process the bipartite graph $(u, v_u) \in E$ in a streaming fashion; thus there is no need to store the entire bipartite graph in memory. There are $O(k|V|)$ result entries in the output P. As written, the union operation in Line 17 creates the theoretical possibility of $O(k \cdot |V| \cdot |U|)$ space (if every signature is a perfect signature for some k and some sequence). If only a best match (instead of all best matches) is desired, one can replace $(\mathcal{U} \cup \{u\})$ with $\{u\}$ to improve space consumption to $O(k \cdot |V|)$. In either case, the algorithm requires space linear to the size of the output.

ALGORITHM: Partial Group Coverage With Outgroup Hits

```
Input: $G(U,V,E)$, $T$, $LCA(v,v')$, $k$
Output: $P : (T, \{0,\dots,k\}) \to (\mathcal{P}(U), \mathbb{N})$
1  for $u \in U$ do
2      for $o \in \{0,\dots,k\}$ do
3          $P(u,o) = (\emptyset, 0)$;
4      end
5  end
6  for $u \in U$ do
7      $S_u \leftarrow$ sort $\{v_u | (u,v_u) \in E\}$;
8      $v_u^{min} \leftarrow$ min $S_u$ ;
9      $v_u^{max} \leftarrow$ max $S_u$ ;
10     $\hat{u} \leftarrow LCA(v_u^{min}, v_u^{max})$;
11     $(\mathcal{U}, n') \leftarrow P(\hat{u}, 0)$;
12     $n \leftarrow |S_u|$;
13     if $n > n'$ then
14         $P(\hat{u}, 0) \leftarrow (\{u\}, n)$;
15     end
16     if $n = n'$ then
17         $P(\hat{u}, 0) \leftarrow (\mathcal{U} \cup \{u\}, n')$;
18     end
19     PropagateDown $(T, \hat{u}, P, S_u, k, u)$ ;
20 end
21 PropagateUpWithK $(T, r, P, k)$;
```

matched by those signatures in the target group. The Procedure PropagateUpWithK corresponds to the procedure PropagateUp, differing only in propagating $k+1$ matches up.

Algorithm 3 requires another helper Procedure PropagateDown which propagates signatures down the tree T. At the LCA node, a signature will only have ingroup matches and it has the highest sensitivity. But the signature could also be valuable candidate for its children, which in their case results in outgroup matches. Propagating signatures downwards requires recalculating the number of ingroup and outgroup hits at each step. Given that at most k outgroup hits are allowed, the propagation stops after a total of at most $O(k)$ downward steps.[3]

At each step, the procedure uses the border B and a binary search to quickly determine the number of outgroup and ingroup hits for the new target group. Let indexof$_{min}(e, S)$ be a function that returns the index of element $e \in S$ in the zero-indexed, sorted array S. If $e \notin S$, the index of the element left of the position e would

[3]Without loss of generality, we can assume that T is a binary tree, thus $|C(p)| \leq 2$ can be assumed and the first iteration over $|C(p)|$ children of p is also in $O(k)$ time.

Procedure PropagateUpWithK(T, p, P, k) Helper function to propagate good matches up the tree using depth first traversal (in $O(k|V|)$ time). This procedure closely corresponds to Procedure PropagateUp, except that we need to propagate the best signatures up $k+1$ times.

```
Input: $T$, $p$, $P : T \to (\mathcal{P}(U), \mathbb{N})$, $k$
Output: Updated $P : T \to (\mathcal{P}(U), \mathbb{N})$
foreach $c \in C(p)$ do
    foreach $o \in \{0, \ldots, k\}$ do
        PropagateUpWithK ($T$, $c$, $P$, $k$);
        if $p \neq r$ then
            $p' \leftarrow$ parent ($p$);
            $(\mathcal{U}, n) \leftarrow P(p, o)$;
            $(\mathcal{U}', n') \leftarrow P(p', o)$;
            if $n > n'$ then
                $P(p', o) \leftarrow (\mathcal{U}, n)$;
            end
            if $n = n'$ then
                $P(p', o) \leftarrow (\mathcal{U} \cup \mathcal{U}', n')$;
            end
        end
    end
end
```

have had in S should be returned (we never use indexof$_{min}(e, S)$ on elements e that have no smaller element in S). Similarly, let indexof$_{max}(e, S)$ be a function that returns the index of element $e \in S$ in the zero-indexed, sorted array S, and if $e \notin S$ returns the index of the element right of the position e would have had in S (again, we never use indexof$_{max}(e, S)$ on elements e that have no larger element in S). Figure 6 illustrates how indexof is used to quickly determine the number of outgroup hits o. As S is sorted, both indexof computations can be done in $O(\log |S|)$ time using binary search. Procedure PropagateDown then lists the steps necessary to propagate signatures down the tree.

Figure 5 illustrates the downwards propagation for $k = 1$. Going towards the left, the propagation immediately terminates as $o = 2 > k$. Propagating towards x, the propagation first updates the result set for $o = 1$ outgroup hit and then terminates on the next level as the number of outgroup hits again rises to $o = 2 > k$.

4. EXPERIMENTAL RESULTS

We implemented the CaSSiS-LCA algorithm from Section 3.3 in C++. The CaSSiS tools and the source code are available as free software (LGPL) at `http://cassis.in.tum.de/`. The website also contains supplementary material, e.g. test datasets, probes mentioned in Section 2 and more detailed instructions.

All experiments were performed on an Intel Core i7-920 (2.67 GHz) Debian GNU/Linux system with 16 GB of main memory. We evaluated the performance of our algorithm using the sequence data and phylogenetic tree from the SILVA SSU-Ref 108[4] dataset. It should be noted that the output of the new algorithm is identical to the output from CaSSiS-BGRT [Bader et al. 2011]. A detailed analysis of the output can be obtained from the CaSSiS website.

[4]`http://www.arb-silva.de/documentation/background/release-108/`

Procedure PropagateDown(T, p, P, S, k, u) Helper function to propagate good matches down the tree, bounded by k. The recursion is bounded to at most $O(k)$ calls, thus the complexity of this procedure is $O(k \log |S|)$ time and thus $O(k \log |V|)$ as $S \subseteq V$.

```
Input: $T$, $p$, $P : T \to (\mathcal{P}(U), \mathbb{N})$, $S$, $k$, $u$, indexof($e, S$)
Output: Updated $P : T \to (\mathcal{P}(U), \mathbb{N})$
foreach $c \in C(p)$ do
    $(v_c^{min}, v_c^{max}) \leftarrow B(c)$;
    $i_c^{min} \leftarrow \text{indexof}_{min}(v_c^{min}, S)$ ;
    $i_c^{max} \leftarrow \text{indexof}_{max}(v_c^{max}, S)$ ;
    $o \leftarrow i_c^{min} + |S| - i_c^{max} - 1$;
    $n \leftarrow |S| - o$;
    if $o \leq k$ then
        $(\mathcal{U}', n') \leftarrow P(c, o)$;
        if $n > n'$ then
            $P(c, o) \leftarrow (\{u\}, n)$;
        end
        if $n = n'$ then
            $P(c, o) \leftarrow (\mathcal{U}' \cup \{u\}, n')$;
        end
        PropagateDown $(T, c, P, S, k, u)$;
    end
end
```

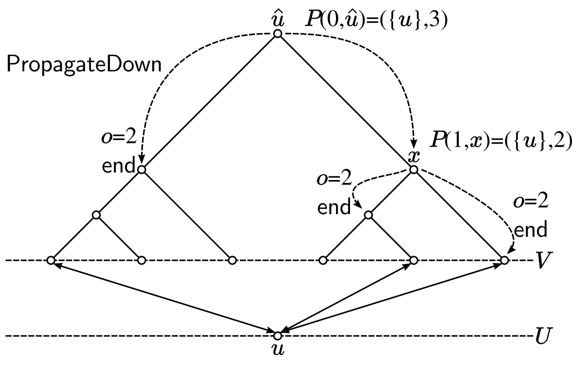


Fig. 5: Illustration of Algorithm 3. For the example, we use an outgroup limit of $k := 1$. At $\hat{u}$ (and its parent nodes) no outgroup matches are possible, i.e. $P(0, \hat{u}) \leftarrow (\mathcal{U}, 3)$. The match is then propagated towards the child nodes. At node x, the match is added with 1 outgroup match, i.e. $P(1, x) \leftarrow (\mathcal{U}, 2)$. In the other child nodes, the outgroup limit k is exceeded.

4.1. Implementation

The inputs to our implementation are a (binary) phylogenetic tree in the Newick format [Olsen 1990] and MultiFasta[5] formatted 16S rRNA gene sequence datasets. Each sequence represents an organism. For our experiments, we used a modified ARB PT-Server to generate the bipartite graph that maps signature candidates to organisms. The modifications allowed direct access to the result sets without double parsing (once

[5]`http://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml`

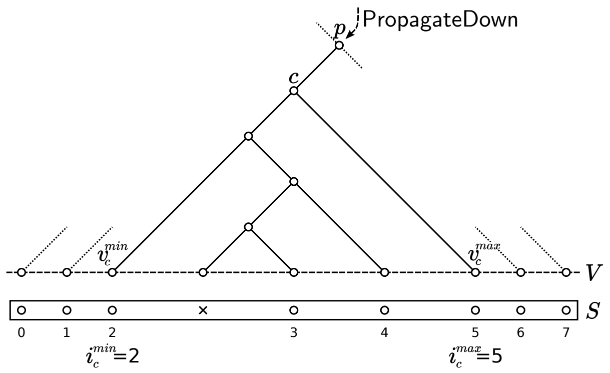


Fig. 6: Detailed illustration of one iteration in the foreach loop of the PropagateDown procedure. The left- and rightmost descendants (v_c^{min} and v_c^{max}) of the node c are read. Then, the indexof procedures are used to fetch their positions in the match set S: $i_c^{min} = 2$ and $i_c^{max} = 5$. The number of outgroup matches at c is $o = i_c^{min} + |S| - i_c^{max} - 1 = 2 + 8 - 5 - 1 = 4$, the number of matches therefore $n = |S| - o = 4$

in the PT-Server and again in CaSSiS). Also, the memory management was adapted to reduce the memory consumption.

The PT-Server allows the definition of a Hamming distance when matching a signature against the sequences. Our implementation utilizes this to allow a certain Hamming distance m_1 for matches within the target group as well as enforcing a minimum Hamming distance $m_2 > m_1$ to sequences outside of the target group. The latter is implemented by adding the number of organisms with a distance between m_1 and m_2 to the initial number of outgroup hits for the probe. This strategy was also used and discussed in [Bader et al. 2011]. In both approaches, using the same Hamming distance values result in identical bipartite graphs.

For the precomputation, we used the canonical approach of reducing the LCA problem into a Range Minimum Query (RMQ) problem [Gabow et al. 1984]. We used the *Sparse Table (ST)* algorithm described by Bender and Farach-Colton [Bender and Farach-Colton 2000] to preprocess with a complexity of $O(n \log n)$ time and to achieve $O(1)$ time for the RMQ queries during the main phase of the algorithm. While solutions for pre-processing the LCA lookup with linear runtime and memory complexity exist [Bender and Farach-Colton 2000; Fischer and Heun 2006; 2007], we did not implement those as the $O(n \log n)$ processing is not the bottleneck (costing less than 1% of the total execution time) and the constant factors for the main phase of the algorithm would be higher for those other schemes. However, for the complexity analysis, we assume that the best-known linear algorithm for LCA could be used and thus the LCA precomputation will take $O(|V|)$ time and space.

4.2. Measurements

We created test datasets of increasing sizes by randomly selecting sequences from the original SILVA SSURef 108 dataset. We reduced the phylogenetic trees to only include leaves which reference the selected sequences. The test datasets range from $16,000$ to $512,000$ sequences (Table II).

As expected, the main phase of the algorithm spends most of its time performing the downward propagation. However, execution time is dominated by far by the queries to the ARB PT-Server (Figure 7), especially for larger Hamming distance values (m_1 and m_2; Table III). It should be noted that it would be trivial to run multiple instances

Sequences $\|V\|$	Bigraph Edges $\|E\|$	Signatures $\|U\|$	Nucleotides
16,000	22,675,424	3,035,608	22,961,088
32,000	45,332,247	4,654,334	45,882,367
64,000	90,712,819	7,110,160	91,766,991
128,000	181,546,054	10,767,681	183,567,793
256,000	363,195,618	16,219,346	367,064,051
512,000	726,690,069	24,125,196	734,101,089
(*) 618,442	882,460,379	31,976,771	891,481,250

Table II: Test datasets derived from the SSURef 108 (*) dataset. Sequences represent rRNA genes of organisms. For each dataset, the total number of nucleotides and unique exact-matching 18 mer signatures are shown. The bigraph edges represent matches between 18 mer signatures and sequences.

of the ARB PT-Server in a cluster [Bader et al. 2010] (memory per node permitting). Distributing the queries across multiple search indices and aggregating the results afterwards would further decrease the runtime of our approach.

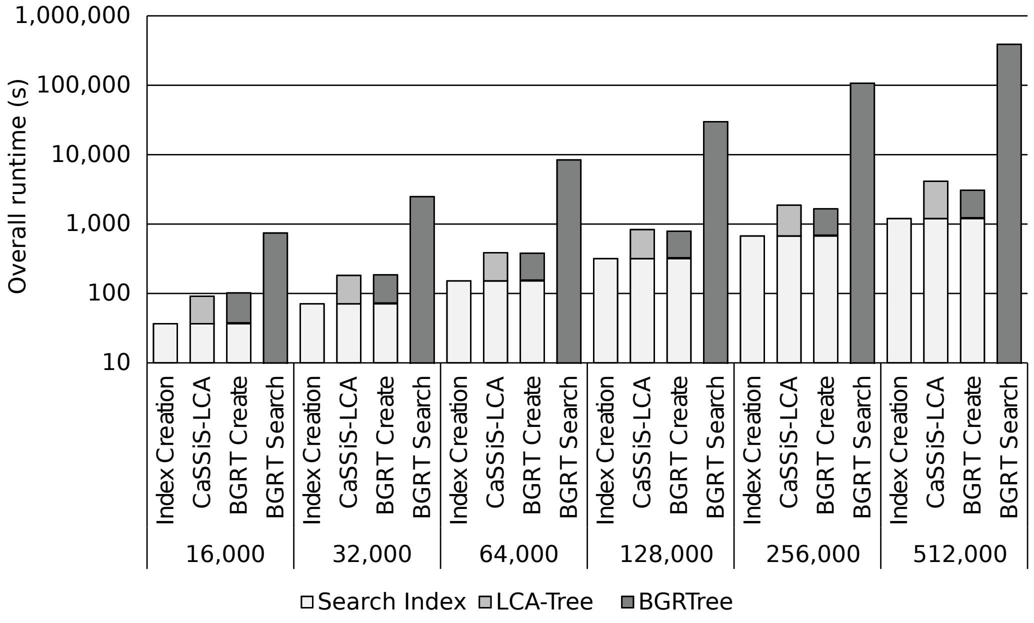


Fig. 7: Overall runtime of the CaSSiS-BGRT and CaSSiS-LCA implementations for growing dataset sizes ($16,000$–$512,000$ sequences) with with $m_1 = 0$, $m_2 = 1$ and $k = 10$. CaSSiS-BGRT was split in its two computing stages BGRT Create and BGRT Search. The creation of the BGRT and the CaSSiS-LCA approach runtimes include building the search index. The measured runtimes grow linear with the dataset sizes.

Both algorithms, CaSSiS-BGRT and CaSSiS-LCA, begin with the computation of the search index. Afterwards, the search index is loaded into memory for further processing. Up to this point runtime and memory consumption for both algorithms are identical. Afterwards, either a BGRT is created and in a later step processed (CaSSiS-BGRT), or the results are directly inserted at the appropriate node in the phylogenetic

Distances m_1	m_2	Runtime (secs)	Edges $\lvert E\rvert$	Signatures $\lvert U\rvert$
0	1	54	22,675,424	3,035,608
0	2	188	3,669,142	1,932,039
0	3	1,167	1,746,651	1,113,679
0	4	5,292	752,918	534,838
1	2	259	369,546,279	3,035,608
1	3	1,233	4,399,485	1,229,004
1	4	5,378	1,315,221	566,610
2	3	1,488	1,386,742,230	3,035,608
2	4	5,628	3,402,659	646,788
3	4	6,203	3,349,697,538	3,035,608

Table III: Allowing mismatches within the target group by increasing the Hamming distance m_1 increases the number of edges $|E|$ in the bipartite graph. The number of signatures $|U|$ decreases with growing Hamming distances m_2 to non-targets. The number of sequences, here $|V| = 16,000$, stays constant. The total runtime is mainly influenced by the time needed to process the search index queries, and therefore increases with growing mismatch distances (m_2).

tree (CaSSiS-LCA). Figure 8 illustrates the growth in memory requirement of all major steps for different input sizes. With growing dataset sizes, the signature search based on the BGRT becomes the most memory consuming step.

We used the largest test dataset to provide a detailed comparison of the two approaches (Figure 9). Due to the identical search index computation and result processing in both implementations, the most significant difference is the runtime of the search for promising signatures at the BGRT- and the LCA-Tree-traversal steps.

5. CONCLUSIONS

We have presented a runtime and memory efficient solution for the the *in silico* search for promising signature candidates even under relaxed search conditions. Combined with modern algorithms for genome-scale pattern matching this approach will allow the computation of specific and sensitive primers and probes.

While the resulting primers and probes are optimal from a computational point of view, future work should also consider biochemical properties such as melting points. The current CaSSiS implementation already contains appropriate filters. Another line for future work will be the search for combinations of multiple probes to cover all organisms in a target group, as for some groups of organisms probes with perfect coverage simply do not exist.

While CaSSiS-BGRT and the CaSSiS-LCA solution presented in this paper are based on the ARB PT-Server, it should be easy to switch to a different search index for pattern matching, such as PtPan [Eißler et al. 2011] or SeqAn [Doring et al. 2008]. PtPan supports Levenshtein distances (the PT-Server currently only supports Hamming distances), which might lead to better results during relaxed searches. SeqAn provides *suffix array* search structures which are considered more memory efficient than suffix tree structures [Abouelhoda et al. 2004]. Both approaches are slower in answering queries when compared to the ARB PT-Server [Eißler et al. 2011].

By partitioning the search index in a cluster, we predict that CaSSiS-LCA should be able to process genome data of virtually arbitrary size. It would mostly be limited by the size of the clustering which is used to store the resulting signature candidates. First promising tests were carried out in a previous work [Bader et al. 2010].

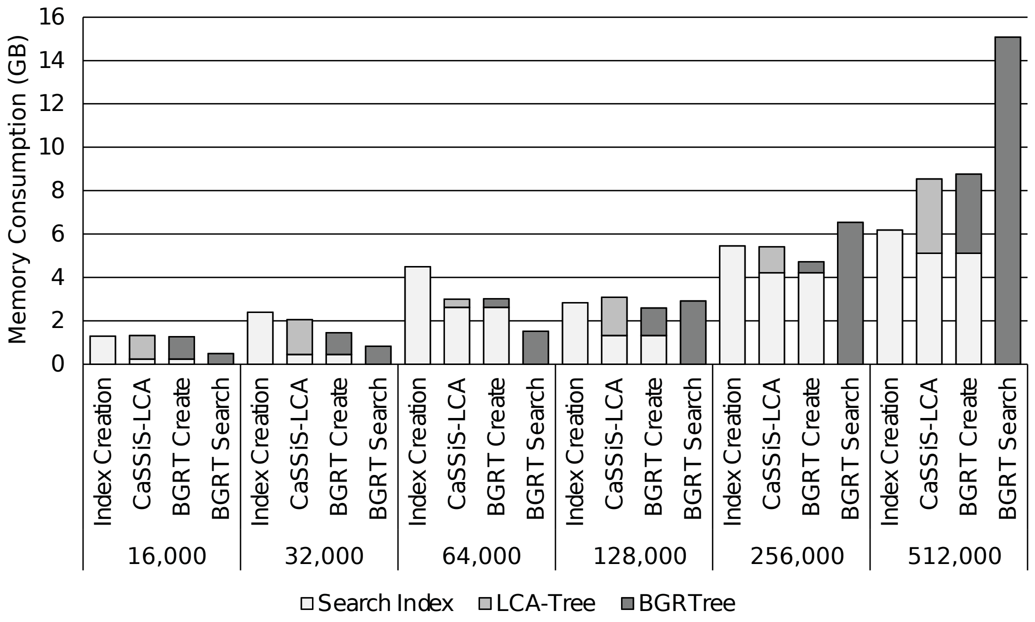


Fig. 8: Comparison of the memory consumption of CaSSiS-BGRT and CaSSiS-LCA for growing dataset sizes ($16,000$–$512,000$ sequences) computed with $m_1 = 0$, $m_2 = 1$ and $k = 10$. Results for the complete SSURef 108 dataset are not shown as the BGRT search step exceeded the available main memory on our test system (20.1 GB; the CaSSiS-LCA implementation only required 10.8 GB). Note that the CaSSiS-BGRT approach was split into its two main steps BGRT Create and BGRT Search. The memory consumption of the search index after its computation is identical for CaSSiS-LCA and BGRT Create. For the BGRT search/traversal the search index is not needed anymore. Traversing the BGRT consumes far more memory than the LCA approach, although the same phylogenetic tree structure is used to store the results (in the nodes).

REFERENCES

ABOUELHODA, M. I., KURTZ, S., AND OHLEBUSCH, E. 2004. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms 2,* 1, 53 – 86. The 9th International Symposium on String Processing and Information Retrieval.

AMANN, R. AND FUCHS, B. M. 2008. Single-cell identification in microbial communities by improved fluorescence in situ hybridization techniques. *Nat. Rev. Microbiol. 6,* 5, 339–348.

AMANN, R. I., LUDWIG, W., AND SCHLEIFER, K. H. 1995. Phylogenetic identification and in situ detection of individual microbial cells without cultivation. *Microbiol Rev. 59,* 1, 143–169.

BADER, K. C., EISSLER, T., EVANS, N., GAUTHIERDICKEY, C., GROTHOFF, C., GROTHOFF, K., KEENE, J., MEIER, H., RITZDORF, C., AND RUTHERFORD, M. J. 2010. Distributed stream processing with DUP. In *Network and Parallel Computing*. Lecture Notes in Computer Science Series, vol. 6289. Springer Berlin / Heidelberg, 232–246.

BADER, K. C., GROTHOFF, C., AND MEIER, H. 2011. Comprehensive and relaxed search for oligonucleotide signatures in hierarchically clustered sequence datasets. *Bioinformatics 27*, 1546–1554.

BARTLETT, J. M. S. AND STIRLING, D. 2003. A short history of the polymerase chain reaction. In *PCR Protocols*, J. M. Bartlett and D. Stirling, Eds. Methods in Molecular Biology Series, vol. 226. Humana Press, 3–6. 10.1385/1-59259-384-4:3.

BENDER, M. AND FARACH-COLTON, M. 2000. The lca problem revisited. In *LATIN 2000: Theoretical Informatics*, G. Gonnet and A. Viola, Eds. Lecture Notes in Computer Science Series, vol. 1776. Springer Berlin / Heidelberg, 88–94.

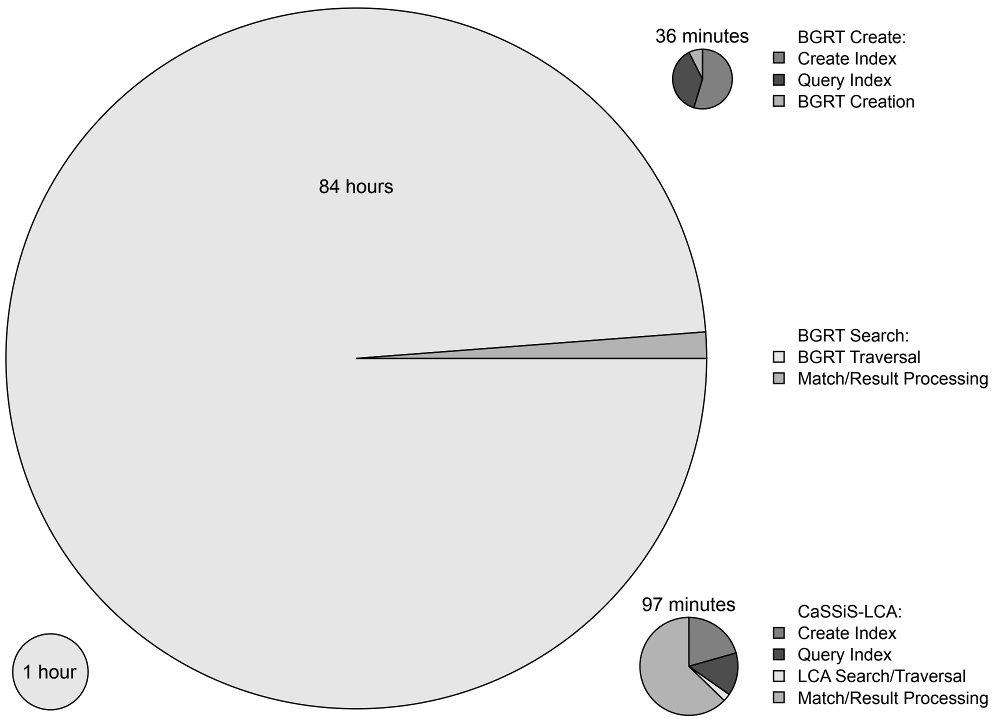


Fig. 9: Detailed comparison of the runtime of CaSSiS-BGRT and CaSSiS-LCA algorithm for the dataset with $512,000$ sequences. CaSSiS-BGRT was split in its two computing stages BGRT Create and BGRT Search. Steps with a runtime below one minute (e.g. loading the BGRT) are not shown. The runtime is represented by area. The most notable difference is the reduction of the 83 hours of the BGRT Traversal step to 148 seconds during LCA Search/Traversal".

CHUNG, W. H., RHEE, S. K., WAN, X. F., BAE, J. W., QUAN, Z. X., AND PARK, Y. H. 2005. Design of long oligonucleotide probes for functional gene detection in a microbial community. *Bioinformatics 21*, 4092–4100.

COLE, J. R., WANG, Q., CARDENAS, E., FISH, J., CHAI, B., FARRIS, R. J., KULAM-SYED-MOHIDEEN, A. S., MCGARRELL, D. M., MARSH, T., GARRITY, G. M., AND TIEDJE, J. M. 2009. The Ribosomal Database Project: improved alignments and new tools for rRNA analysis. *Nucleic Acids Res. 37*, D141–145.

DESANTIS, T. Z., HUGENHOLTZ, P., LARSEN, N., ROJAS, M., BRODIE, E. L., KELLER, K., HUBER, T., DALEVI, D., HU, P., AND ANDERSEN, G. L. 2006. Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Appl. Environ. Micorbiol. 72,* 7, 5069–5072.

DORING, A., WEESE, D., RAUSCH, T., AND REINERT, K. 2008. Seqan an efficient, generic c++ library for sequence analysis. *BMC Bioinformatics 9,* 1, 11.

DUITAMA, J., KUMAR, D. M., HEMPHILL, E., KHAN, M., MANDOIU, I. I., AND NELSON, C. E. 2009. PrimerHunter: a primer design tool for PCR-based virus subtype identification. *Nucleic Acids Res. 37*, 2483–2492.

EISSLER, T., HODGES, C. P., AND MEIER, H. 2011. Ptpan — overcoming memory limitations in oligonucleotide string matching for primer/probe design. *Bioinformatics 27,* 20, 2797–2805.

FENG, S. AND TILLIER, E. R. M. R. 2007. A fast and flexible approach to oligonucleotide probe design for genomes and gene families. *Bioinformatics 23*, 1195–1202.

FICETOLA, G. F., COISSAC, E., ZUNDEL, S., RIAZ, T., SHEHZAD, W., BESSIERE, J., TABERLET, P., AND POMPANON, F. 2010. An in silico approach for the evaluation of DNA barcodes. *BMC Genomics 11*, 434.

FISCHER, J. AND HEUN, V. 2006. Theoretical and practical improvements on the rmq-problem, with applications to lca and lce. In *Combinatorial Pattern Matching*, M. Lewenstein and G. Valiente, Eds. Lecture Notes in Computer Science Series, vol. 4009. Springer Berlin / Heidelberg, 36–48.

FISCHER, J. AND HEUN, V. 2007. A new succinct representation of rmq-information and improvements in the enhanced suffix array. In *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, B. Chen, M. Paterson, and G. Zhang, Eds. Lecture Notes in Computer Science Series, vol. 4614. Springer Berlin / Heidelberg, 459–470.

GABOW, H. N., BENTLEY, J. L., AND TARJAN, R. E. 1984. Scaling and related techniques for geometry problems. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. STOC '84. ACM, New York, NY, USA, 135–143.

KURTZ, S., PHILLIPPY, A., DELCHER, A. L., SMOOT, M., SHUMWAY, M., ANTONESCU, C., AND SALZBERG, S. L. 2004. Versatile and open software for comparing large genomes. *Genome Biol. 5*, R12.

LEE, H. P., SHEU, T.-F., AND TANG, C. Y. 2010. A parallel and incremental algorithm for efficient unique signature discovery on dna databases. *BMC Bioinformatics 11*, 132.

LOY, A., MAIXNER, F., WAGNER, M., AND HORN, M. 2007. probeBase–an online resource for rRNA-targeted oligonucleotide probes: new features 2007. *Nucleic Acids Res. 35,* Database issue, D800–804.

LUDWIG, W., STRUNK, O., WESTRAM, R., RICHTER, L., MEIER, H., YADHUKUMAR, BUCHNER, A., LAI, T., STEPPI, S., JOBB, G., FÖRSTER, W., BRETTSKE, I., GERBER, S., GINHART, A. W., GROSS, O., GRUMANN, S., HERMANN, S., JOST, R., KÖNIG, A., LISS, T., LÜSSMANN, R., MAY, M., NONHOFF, B., REICHEL, B., STREHLOW, R., STAMATAKIS, A., STUCKMANN, N., VILBIG, A., LENKE, M., LUDWIG, T., BODE, A., AND SCHLEIFER, K. H. 2004. ARB: a software environment for sequence data. *Nucleic Acids Res. 32,* 4, 1363–1371.

OLSEN, G. 1990. The "Newick's 8:45" tree format. `http://evolution.genetics.washington.edu/phylip/newick_doc.html`.

PETERLONGO, P., NICOLAS, J., LAVENIER, D., VORCH, R., AND QUERELLOU, J. 2009. c-gamma:comparative genome analysis of molecular markers. In *Pattern Recognition in Bioinformatics*, V. Kadirkamanathan, G. Sanguinetti, M. Girolami, M. Niranjan, and J. Noirel, Eds. Lecture Notes in Computer Science Series, vol. 5780. Springer Berlin / Heidelberg, 255–269.

PHILLIPPY, A. M., AYANBULE, K., EDWARDS, N. J., AND SALZBERG, S. L. 2009. Insignia: a DNA signature search web server for diagnostic assay development. *Nucleic Acids Res. 37*, W229–234.

PHILLIPPY, A. M., MASON, J. A., AYANBULE, K., SOMMER, D. D., TAVIANI, E., HUQ, A., COLWELL, R. R., KNIGHT, I. T., AND SALZBERG, S. L. 2007. Comprehensive DNA signature discovery and validation. *PLoS Comput. Biol. 3*, e98.

PRICE, M. N., DEHAL, P. S., AND ARKIN, A. P. 2010. Fasttree 2 approximately maximum-likelihood trees for large alignments. *PLoS ONE 5,* 3, e9490.

PRUESSE, E., QUAST, C., KNITTEL, K., FUCHS, B. M., LUDWIG, W., PEPLIES, J., AND GLÖCKNER, F. O. 2007. SILVA: a comprehensive online resource for quality checked and aligned ribosomal RNA sequence data compatible with ARB. *Nucleic Acids Res. 35,* 21, 7188–7196.

RIAZ, T., SHEHZAD, W., VIARI, A., POMPANON, F., TABERLET, P., AND COISSAC, E. 2011. ecoPrimers: inference of new DNA barcode markers from whole genome sequence analysis. *Nucleic Acids Res. 39*, e145.

STAHL, D. A., FLESHER, B., MANSFIELD, H. R., AND MONTGOMERY, L. 1988. Use of phylogenetically based hybridization probes for studies of ruminal microbial ecology. *Appl. Environ. Microbiol. 54*, 1079–1084.

STAMATAKIS, A. 2006. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics 22*, 2688–2690.

Received January 2012; revised April 2012; accepted April 2012